SKYLINE BASED TERRAIN MATCHING

NAGW-1333

by

Lance A. Page

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering
Troy, New York 12180-3590

December, 1990

CIRSSE REPORT #79

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I wish to express my appreciation to C. N. Shen, for the time and confidence he invested in me as this work progressed.

Special acknowledgement also to the staff and fellow students of CIRSSE, who have made it an enjoyable place to work.

Finally I express thanks to my mother Loraine and stepfather George, for their love and unflinching support through these years.

ABSTRACT

This thesis describes skyline-based terrain matching, a new method for locating the vantage point of stereo camera or laser range-finding measurements on a global map previously prepared by satellite or aerial mapping. The orientation of the vantage is assumed known, but its translational parameters are determined by the algorithm.

Skylines, or occluding contours, can be extracted from the sensory measurements taken by an autonomous vehicle. They can also be modelled from the global map, given a vantage estimate to start from. The two sets of skylines, represented in cylindrical coordinates about either the true or the estimated vantage, are employed as "features" or reference objects common to both sources of information. The terrain matching problem is formulated in terms of finding a translation between the respective representations of the skylines. by approximating the two sets skylines as identical features (curves) on the actual terrain. The search for this translation is based on selecting the longest of the minimum-distance vectors between cooresponding curves from the two sets of skylines. In successive iterations of the algorithm, the approximation that the two sets of curves are identical becomes more accurate, and the vantage estimate continues to improve.

The algorithm has been implemented and evaluated on a simulated terrain. Illustrations and examples are included.

CHAPTER 1
INTRODUCTION AND LITERATURE REVIEW

The task of *terrain matching* is to check or refine an autonomous vehicles's estimate of its own *vantage*, that is, position and orientation of its visual sensors with respect to a "global map" of the terrain. Terrain matching is one of the basic subtasks of autonomous vehicle navigation. Once the vantage is established, the vehicle may perform path selection based on its current position, the information from both the global map and its sensors, and whatever instructions have been issued to the vehicle. It may then proceed to move along the selected path segment by dead reckoning, to a new position from which new sensor measurements may be taken.

Terrain matching is an important step in the autonomous vehicle's navigation primarily because errors in dead reckoning, for instance due to slipping, must not be allowed to accumulate. If the position errors did accumulate, then the difference between where the vehicle "thought" it was and where it actually was would eventually grow until the vehicle was irrecoverably lost. Terrain matching corrects such errors before they get out of control.

We have developed a new method for performing the terrain matching, suitable for semi-autonomous navigation over a $2\frac{1}{2}$-D unstructured, partially known terrain. "$2\frac{1}{2}$-D unstructured" refers to a terrain on which the ground height is an arbitrary, continuous function of horizontal position. Caves or overhangs violate such a terrain model. "Partially known" means that we are provided a rough, *global map* of the terrain ahead of time. The case of the Mars Rover, on the Mars Sample Return Mission is one such application[1, 2]. In this case, the global map will consist of terrain height samples based on satellite observations made before the Rover's deployment.

The new technique, called skyline-based terrain matching, uses *skylines* as features which are common to both maps in order to determine the error in the current vantage estimate. "Local skylines" are the skylines, or visual occluding edges, which the vehicle "sees" with its three-dimensional (3-D) sensors. They can be generated simultaneously with the local map, and in fact mark the edges between visible and hidden regions of the terrain. "Global skylines" are the curves that the vehicle expects to see, based on its expected or *a priori* vantage estimate, and the global map. Each iteration of the algorithm generates a set of global skylines for the most recent vantage estimate, compares them to the local skylines, and produces a new vantage estimate.

The vantage has six parameters: three for translation, and three for orientation (yaw, pitch, and roll). In the current formulation of the terrain matching algorithm, all three orientation parameters are assumed known. For simplicity, we assume that the vehicle's pitch and roll are zero.

We proceed with the introduction by describing the context in which our development was originally formulated: the Mars Rover Sample Return (MRSR) Mission; next we outline a whole semi-autonomous navigation paradigm, suitable for the MRSR application: we overview some of the alternative approaches to terrain matching and related problems: and we conclude the introduction by clearing up the concept of skylines.

The remainder of the thesis is presented as follows: Chapter 2 develops the terrain and map models on which the terrain matching algorithm has been tested and evaluated; Chapter 3 describes a basic component of skyline-based terrain matching: the method for generating skylines from the global map. Chapter 4 describes the specific formulation and implementation of the skyline-based terrain matching algorithm, except for one part of the algorithm, called "curve correspondence," which has not been developed: Chapter 5 presents results of simulations of the algorithm.

including experiments which illustrate each main step of the terrain matching algorithm, as well as several which demonstrate the whole algorithm at work; Chapter 6 concludes the thesis, with a discussion of the results and suggestions for future work. To a large degree, this thesis brings together the work presented in the three papers [3, 4, 5], with some details and results heretofore unpublished.

1.1 Mars Rover Sample Return (MRSR) Mission

The Mars Sample Return Mission demands a roving vehicle capable of traveling to different locations, extracting surface samples (dirt and rocks), and returning them to "pristine" chambers in a Mars Lander which will carry the samples to Earth. The robot will remain on Mars to perform reconnaissance and other experiments. Its navigation must be at least semi-autonomous; communication delays between Earth and Mars preclude tele-operating the vehicle to make local navigation decisions during a long distance traversal.

We are developing a semi-autonomous navigation scheme suitable for the Mars Rover's mission. We assume that the robot is equipped with stereo cameras. and/or a laser range-finder. In addition. gross terrain data from previous observations. particularly satellite photographs, are available. A path selection algorithm uses both of these sources of information to choose one segment at a time of a path leading to a predefined destination. In addition. a gross path plan which the path selection algorithm is expected to roughly follow may already be provided.

The satellite observations give a rough *global map* of the terrain surrounding the robot, and the sensory observations provide a *local map*. Each source of information has advantages and limitations. The rough map provides information which may be hidden (occluded) from the sensors, but it is limited in accuracy and resolution; the terrain height samples will likely be on the order of three meters apart[6]. The local map includes much more detail. but is limited to the terrain features that are visible from the robot's vantage point (the part of the robot where the visual sensors are mounted). Additional limitations may apply to the sensory data, depending upon which type of instrument is used.

1.2 Semi-autonomous navigation

Semi-autonomous navigation refers to an unmanned vehicle's ability, given its starting position, its destination, and rough plan to get there, to reach the destination without further instructions. Semi-autonomous navigation typically consist of four main steps, repeated until the destination is reached (adapted from [3, 2, 7]):

1. Sensing the environment in order to build a three-dimensional *local map*, which contains detailed information about a limited, visible region of the terrain.

2. *Terrain matching*, also known as robot localization [8] or position estimation [9, 10], which determines the relationship between the local map and some fixed, global reference. This relationship is expressed as the *vantage*, which is the location and orientation of the vehicle's sensors with respect to some global reference.

3. *Path selection*, which selects the next five to twenty-five meter segment of the journey, based upon mission safety, expediency, fuel conservation, and expected visibility at the next stopping point.

4. Moving the vehicle along the selected path segment.

We shall now elaborate upon the steps listed.

The robot's onboard vision system provides the information necessary to perform path selection according to relatively fine features of the terrain. This information will be in the form of an elevation map (i.e. vertical height as a function of horizontal position), called the "local map," generated from the sensory depth map.

In addition to the elevation map, a set of skylines based on the sensors is produced as a basis for relating the local map to the rough map.

For a depth map from either range finders and stereo cameras, the skylines can be found as discontinuities (or large first differences, for discrete data) in range, once a depth map has been generated. In addition, the cameras can use clues of color and shade changes to detect the skylines[11].

Terrain matching refers to estimating the location and orientation of the robot by comparing the skylines which it was expected to see (based on the rough map) with those that it "actually" sees (with its sensors). The location and orientation of the robot (or more precisely, of its sensors) have a total of six parameters: three for translation (x_v, y_v, and z_v), and three for orientation (Θ_v for azimuth or yaw, β_v for pitch, and Φ_v for roll, i. e. the system III Euler angles [12]). Collectively these are called the *vantage*. $\mathbf{V} = [x_v, y_v, z_v, \Theta_v, \beta_v, \Phi_v]^T$. The kinematic relationship between the mounting of the sensors and the body of the robot then determines the robot's "position."

Based on both maps available to the robot, a path segment is chosen according to the following criteria:

- Amount of progress that the segment makes toward the destination, and/or deviation that it represents from the gross path.
- Inpath and crosspath slopes. Excessive inpath slopes can cause traction to be lost, and energy to be wasted. Excessive crosspath slopes jeopardize the vehicle's lateral stability.
- Obstruction heights (bottom-of-vehicle clearance) along the path.
- Predicted degree of visibility at the next stopping point.

The detailed information in the local map is emphasized in the path selection. Since the rough map is not a reliable source of information for all the possible hazards at any part of the terrain, the robot will not actually proceed to or across any part of the terrain which it cannot observe locally with its own sensors. Nevertheless, the knowledge that the rough map offers about upcoming terrain, however incomplete, may be important for evaluating the desirability of the next stopping point.

The last step in each navigation cycle is to execute the selected path. We assume that dead reckoning control on the robot is capable of moving the robot with reasonable accuracy through the selected path. Position errors will inevitably result from the dead reckoning, but the terrain matching performed at each navigation step prevents the errors from accumulating and growing too large.

1.3 Terrain matching literature review

Slight variants of the problem we call terrain matching have been referred to in the literature under several terms, including scene matching, the pose problem, or robot localization. The crux of the problem is to determine the relative positions of two (or more) sensors, by comparing the data which is provided by the sensors.

The problem arises in a wide variety of applications of autonomous mobile systems. In our case, one set of sensors is on the Mars Rover, and the other is the orbiting array of sensors that were used for generating the global map, and we perform the terrain matching to correct for errors in the dead reckoning path (and in the position estimate of the previous stopping point). Our application is in an unstructured, roughly mapped terrain which lacks distinct "features" such as roads, boulders, signposts, etc., which are perceptible from both the orbital sensors and the onboard sensors: our approach is treat the skylines visible from the onboard sensors as features (or pseudo-features) which can also be constructed from the global map data. In addressing the same problem, Gennery[13] uses an "iconic" approach, matching the entire matrix of data in the local map to the global map, without attempting to reduce the data to "features" ahead of time.

Much investigation has been done on the matching/localization problem in more structured environments, e.g [10, 14]. Lee *etal* [15] proposed the deployment of "indication-posts," to facilitate navigation through unfamiliar environments.

Much work has also been done in the area of fusing successive local maps into one coherent representation, particularly in outdoor, reconnaissance applications, e.g. [16]. Hebert *etal* [11, 17] perform sensory map fusion for the Mars Rover application by employing both feature-based and iconic stages of matching. The advantage in the map-fusion task over the terrain matching task, however, is that the resolutions of the information sources are similar, and thus the features visible (such as boulders) to them are more naturally comparable.

1.4 "What are skylines?"

In the most intuitive terms, skylines are the basic curves which we would draw in a crude depiction of a scene, especially mountains or "rolling hills." such as in Figure 1.1. Skylines are the occluding edges, or local horizons, of the terrain that are visible from some vantage point above the terrain. They are 3-D curves, even though they are characterized by their behavior on the 2-D image projections onto their vantages.

Geometrically, skylines on $2\frac{1}{2}$-D terrain can be defined as the connected sets of points on the terrain where, in terms of spherical coordinates about the vantage point, the following conditions hold: first, that there is a local maximum of elevation angle with respect to radial distance from the vantage; and second, that the point is not obscured by a skyline curve crossing the same azimuth, which is closer to the vantage and has a higher elevation angle.

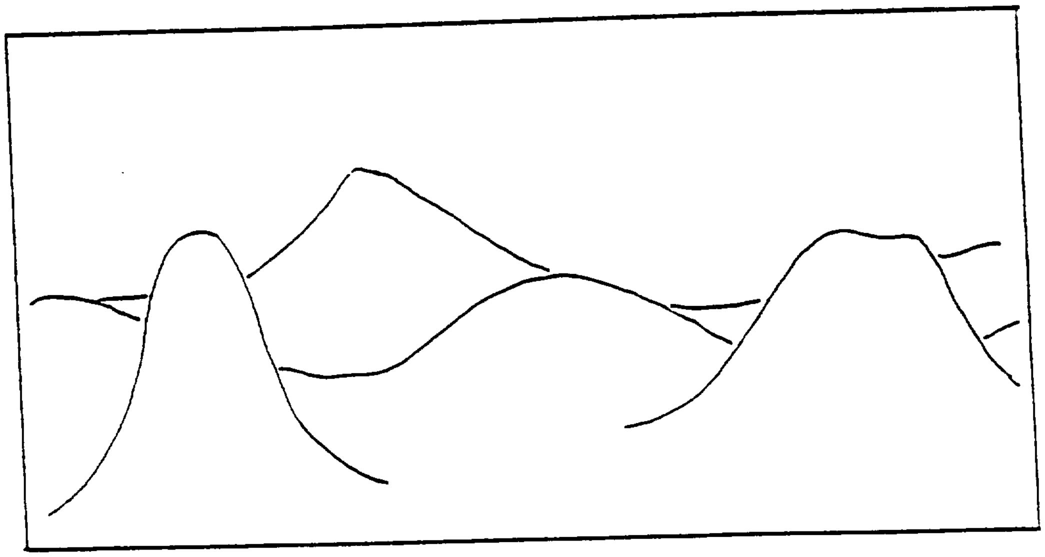
Figure 1.1: Example skylines: "rolling hills" and "bumps."

CHAPTER 2
TERRAIN AND MAP MODELLING

This chapter describes how we have modelled a terrain and a rough map of that terrain, for the simulation and testing of our terrain matching algorithm.

2.1 Coordinate systems

Three main coordinate systems are used:

- Global coordinates. A global, rectangular reference frame, (x^g, y^g, z^g), is fixed to the terrain with the y^g-axis pointing North, and x^g-axis pointing East, and the z^g-axis pointing straight up. (Planetary curvature is not considered in this model.)

- Local coordinates. A local coordinate system, (x^l, y^l, z^l) originates at the vehicle's vantage point, and the relationship between the local and global coordinate systems is a translation:

$$\begin{aligned} x^g &= x_v + x^l \\ y^g &= y_v + y^l \\ z^g &= z_v + z^l \end{aligned} \qquad (2.1)$$

  The vantage point is always assumed be two meters above the terrain. (Note that the local coordinate system does not rotate, even though the vehicle itself may rotate.

- Star coordinates. A coordinate system called the star ($\star$) coordinate system originates at the current vantage estimate, and relates to the global system as

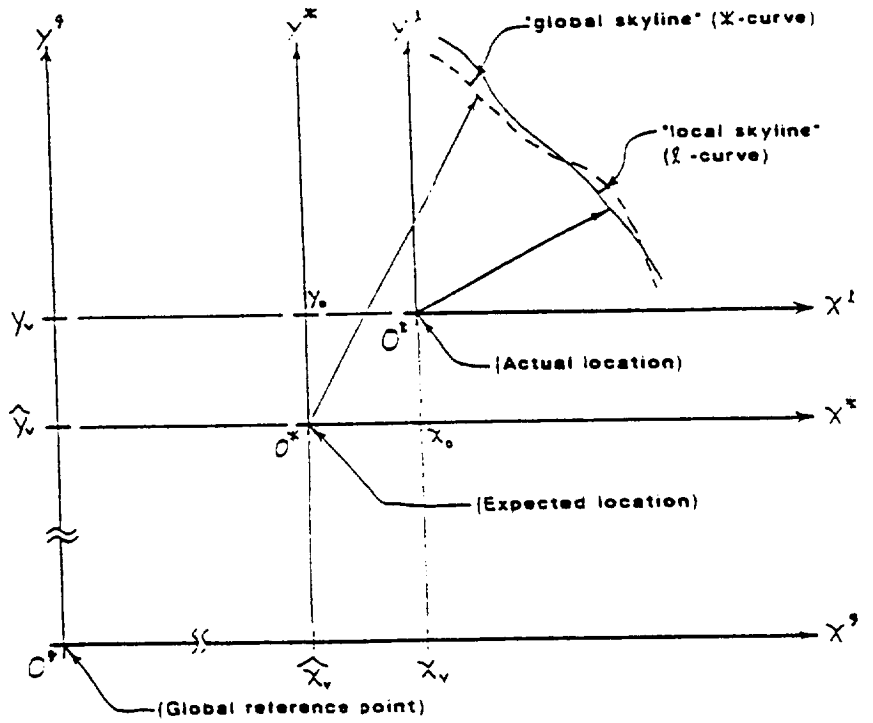

Figure 2.1: Relationships among the global(g), local (l), and star ($\star$) coordinate systems.

follows:

$$\begin{aligned} x^g &= \hat{x}_v + x^* \\ y^g &= \hat{y}_v + y^* \\ z^g &= \hat{z}_v + z^* \end{aligned} \tag{2.2}$$

The relationships among the global. local, and star coordinate systems are illustrated in Figure 2.1. It is emphasized that the transformation from the global to the star coordinate system is known exactly, whereas the transformation to the local coordinate system, fixed to the true vantage. is not.

Throughout this report, each cylindrical coordinate system is related to its corresponding rectangular coordinate system with θ (azimuth) measured counterclockwise from the x-axis; ρ as the length of the horizontal projection: and z the

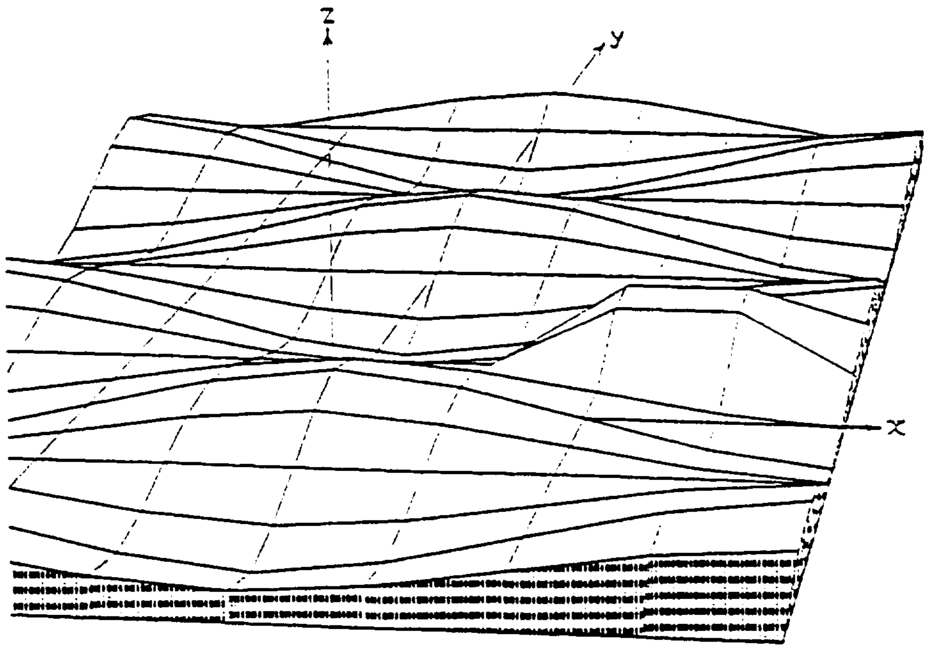

Figure 2.2: Simulated terrain. Grid shown has 5 m by 5 m resolution.

same as the rectangular z. Mathematically,

$$\begin{aligned} \theta &= \text{atan2}(x, y) \\ \rho &= \sqrt{x^2 + y^2} \\ z &= z \end{aligned} \tag{2.3}$$

where the atan2(x, y) function is a special version of $\arctan(y/x)$, which insures that the resulting angle is in the appropriate quadrant.

2.2 Modelling a terrain

The terrain is modelled by a mathematical *terrain function* $z(x, y)$. The terrain function currently used is depicted in Figure 2.2, and consists of an "infinite egg-carton" shape with a monolith superimposed on it to break the monotony.

The underlying, mathematical terrain function is

$$z(x, y) = f(x)g(y) \tag{2.4}$$

where $f(x)$ and $g(y)$ are periodic functions in x and y. $f(x)$ will now be described in detail; the development of $g(y)$ is completely analogous.

The function $f(x)$ is defined as

$$f(x) = [a_0 + a_2(x - x_s)^2 + a_4(x - x_s)^4](-1)^m \tag{2.5}$$

where[1]:

$$m = \left\lfloor \frac{x + x_m}{2x_m} \right\rfloor \tag{2.6}$$

$$x_s = 2mx_m \tag{2.7}$$

and x_m, a_0, a_2, and a_4 are constants described below. Within any domain of $-x_m \le x \le x_m$, $f(x)$ is simply a symmetric, fourth-order polynomial function. Throughout the domain $x \in \Re$, $f(x)$ is C2-continuous and periodic.

The constants a_0, x_m, and x_{slope}represent the peak magnitude of the function, one fourth the period of $f(x)$, and the maximum slope of $f(x)$, respectively. C2-continuity requires that

$$a_2 = -\frac{4a_0 - x_m/x_{slope}}{2x_m^2} \tag{2.8}$$

$$a_4 = \frac{2a_0 - x_m/x_{slope}}{2x_m^4} \tag{2.9}$$

Similarly for function $g(y)$,

$$g(y) = (b_0 + b_2(y - y_s)^2 + b_4(y - y_s)^4)(-1)^n, \tag{2.10}$$

$$n = \left\lfloor \frac{y + y_m}{2y_m} \right\rfloor \tag{2.11}$$

$$y_s = 2ny_m, \tag{2.12}$$

$$b_2 = -\frac{4b_0 - y_m/y_{slope}}{2y_m^2} \tag{2.13}$$

$$b_4 = \frac{2b_0 - y_m/y_{slope}}{2y_m^4} \tag{2.14}$$

[1]Definition: $\lfloor p \rfloor$ equals the greatest integer that is less than or equal to p.

The resulting $z(x,y)$ is periodic in both x and y, with periods of $4x_m$ and $4y_m$. It presents an "infinite egg carton" effect, with alternating hills and depressions along either direction.

The values of ab, x_m, y_m, x_{slope}, and $y_{slope}$are called the *terrain specifications.* Letting

$$a_0 = b_0 = \sqrt{ab} \tag{2.15}$$

Eqs. 2.4–2.15 completely specify $z(x,y)$ in terms of these specifications. In practice, the coefficients a_0, a_2, a_4, b_0, b_2, and b_4 are calculated only once for a set of terrain specifications. To compute a particular $z(x,y)$, $(x - x_s)$ and $(y - y_s)$ along with $(-1)^m$ and $(-1)^n$ are calculated, and then Eqs. 2.4, 2.5, and 2.10 are applied.

The partial derivatives of z with respect to x or y are easily calculated from Eqs. 2.4, 2.5, and 2.10 as follows:

$$\frac{\partial z(x,y)}{\partial x} = g(y)(2a_2(x - x_s) + 4a_4(x - x_s)^3)(-1)^m \tag{2.16}$$

$$\frac{\partial z(x,y)}{\partial y} = f(x)(2b_2(y - y_s) + 4b_4(y - y_s)^3)(-1)^n \tag{2.17}$$

2.3 Modelling a rough map

Intuitively and mathematically, the map is an estimation of the terrain. Our map model provides accurate values of $z_m(x,y)$ only for discrete values of x and y. For any (x,y) that does not happen to be one of these "known" *map points*, $z_m(x,y)$ must be computed based on nearby map points.

Refer to Figure 2.3. The discrete distance between map points in the x and y directions is the *resolution* Δ_m of the map. Every position (x,y), including map points, has a pair of *map indices* (i_m, j_m),

$$i_m = \lfloor x/\Delta_m \rfloor \tag{2.18}$$

$$j_m = \lfloor y/\Delta_m \rfloor \tag{2.19}$$

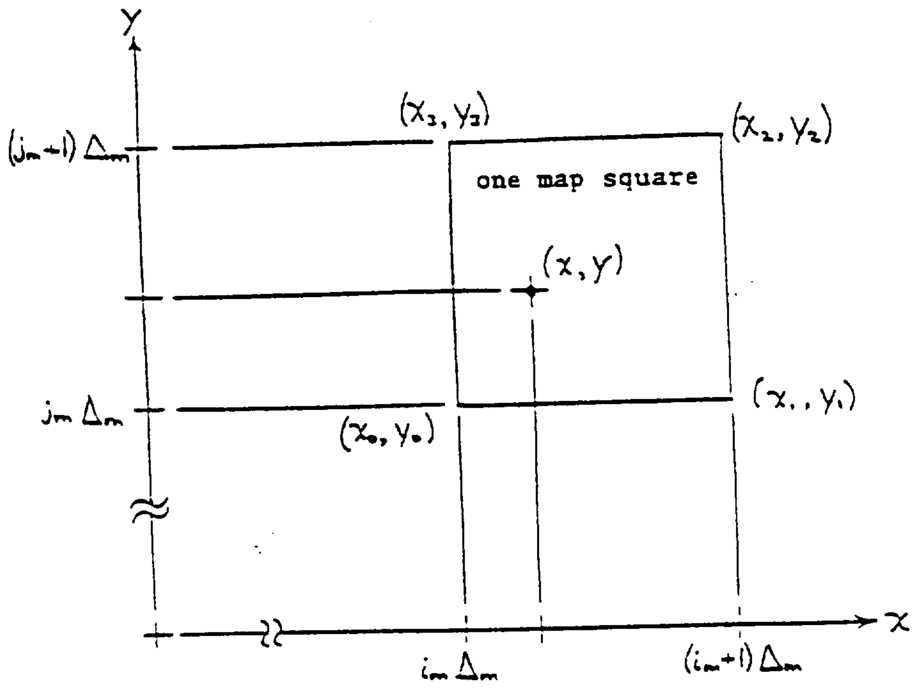

Figure 2.3: Parameters of the Map Model

and as a result,

$$\begin{aligned} (i_m)\Delta_m &\le x < (i_m+1)\Delta_m \\ (j_m)\Delta_m &\le y < (j_m+1)\Delta_m \end{aligned} \tag{2.20}$$

Each pair of map indices thus defines a *map square* in the xy-plane, with its four corners numbered as in Figure 2.3.

2.3.1 Interpolating between map points

The basic interpolation scheme used is simple. For any position (x, y), the value of z_m is interpolated from the known values of z_m at the corner points of the map square which surrounds (x, y), according to the equation:

$$z_m = c_0 + c_1 x + c_2 y + c_3 xy \tag{2.21}$$

to the known values of z_m at the corner points of the map square which contains (x, y).

Assigning $(x_0, y_0) \ldots (x_3, y_3)$ to the corners of the map square, the coefficient vector $\mathbf{c}$ is calculated as

$$\mathbf{c} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \mathbf{A}^{-1}\mathbf{z} \tag{2.22}$$

where

$$\mathbf{A} = \begin{bmatrix} 1 & x_0 & y_0 & x_0y_0 \\ 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_2 & x_2y_2 \\ 1 & x_3 & y_3 & x_3y_3 \end{bmatrix}, \qquad \mathbf{z} = \begin{bmatrix} z(x_0, y_0) \\ z(x_1, y_1) \\ z(x_2, y_2) \\ z(x_3, y_3) \end{bmatrix}$$

2.3.2 Speeding up the interpolation

Two modifications to the basic interpolation scheme make it much more efficient: translating the process, and buffering the coefficient vector.

To eliminate the matrix inversion we apply the following translation:

$$x_r = x - i_m\Delta_m \tag{2.23}$$

$$y_r = y - j_m\Delta_m \tag{2.24}$$

where the origin of the x_r and y_r axes [2] is the 'lower left' corner of the map square. This changes Equation 2.21 to

$$z_m = c_{r0} + c_{r1}x_r + c_{r2}y_r + c_{r3}x_ry_r, \tag{2.25}$$

with solution,

$$\mathbf{c}_r = \mathbf{A}_r^{-1}\mathbf{z}. \tag{2.26}$$

The key to the savings is that the relative corner positions are always the same. For instance, if the four corners of the map square are numbered starting at

[2]The "r" subscript is for "relative."

the x_r-y_r origin and counting clockwise, then

$$\text{For all } (x,y)\text{: } \mathbf{A}_r = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \Delta_m & 0 & 0 \\ 1 & \Delta_m & \Delta_m & {\Delta_m}^2 \\ 1 & 0 & \Delta_m & 0 \end{bmatrix} \tag{2.27}$$

Thus $\mathbf{A}_r^{-1}$ only needs to be computed and stored once— the matrix inversion is effectively eliminated.

Very often, points are "looked up" on the map in groups, and two consecutive map inquiries are made in the same map square (e. g. at increments along a line in Section 4.2). This case occurs so commonly that it is worthwhile to retain the "previous" coefficient vector and map indices. If the map indices of (x,y) match the map indices that the coefficient vector was most recently calculated for, then the old coefficient vector may be used. The steps of looking up four z's from the map and multiplying $\mathbf{A}_r^{-1}$ by $\mathbf{z}$ are eliminated, at the cost of comparing two pairs of integers.

2.3.3 Partial Derivatives

The estimated partial derivatives of z_m with respect to x or y are given as (see Equation 2.25):

$$\frac{\partial z_m(x,y)}{\partial x} = c_{r1} + c_{r3}y \tag{2.28}$$

$$\frac{\partial z_m(x,y)}{\partial y} = c_{r2} + c_{r3}x \tag{2.29}$$

Since these are the same coefficients as the ones used for calculating $z_m(x,y)$, the steps for calculating z_m and its partial derivatives are identical except for the final step of applying Equation 2.25, 2.28, or 2.29.

CHAPTER 3
SKYLINE GENERATION

A set of skylines based on the rough map can be generated for any vantage point above the terrain. (The robot's vantage point is normally assumed to be two meters above the terrain.

This chapter describes how a set of skyline curves visible from a particular vantage point are generated based on information from the rough map.

The skyline curves are generated in three steps: collecting individual ridge points (Sect. 3.1); grouping the points into sets representing curves (Sect. 3.2); and interpolating between the points, i. e. connecting them, for each skyline curve (Sect. 3.3).

3.1 Collecting skyline points

The algorithm for collecting skyline points iterates through a series of azimuths which span a predefined range of directions centered about the vehicle's yaw, and which are separated by equal intervals. Each azimuth is processed by the following steps:

1. "Look up" a series of terrain height samples $z_m(x, y)$ from the rough map, along a line which in the horizontal projection starts at the vantage and proceeds in the direction of the azimuth. This generates a planar cross-section, or *terrain section* of the environment according to the map.

2. Along that section, detect local peaks of *elevation angle* β with respect to horizontal distance ρ. (It can be shown that a maximum in β with respect to ρ is equivalent to a maximum in β with respect to radial distance r.) A local peak is detected when one sample elevation angle is greater than both

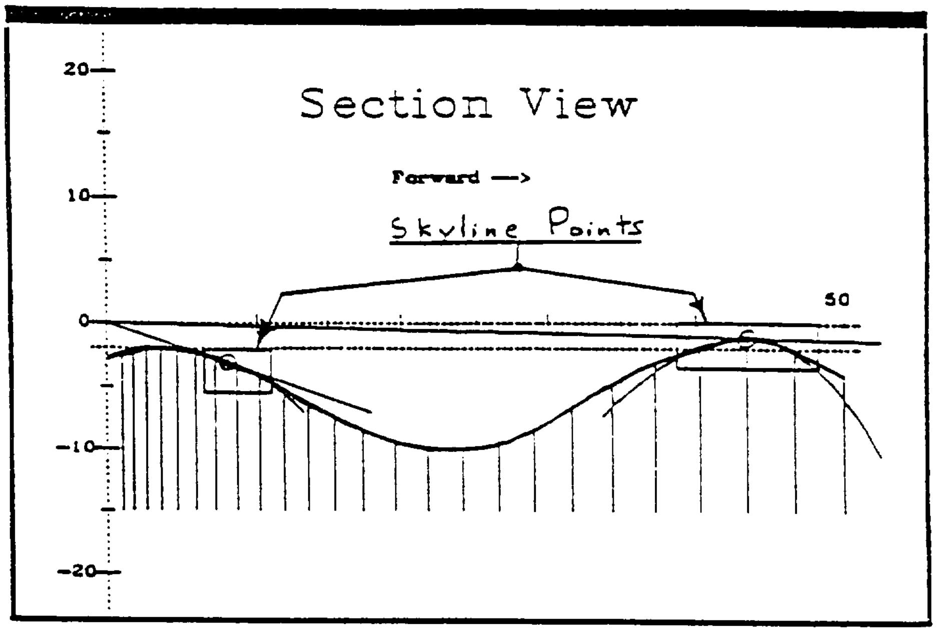

Figure 3.1: Example terrain section, showing two skyline points.

of its neighboring samples. A quadratic function of β vs. ρ is fitted to each of the three-sample sequences, and the maximum of each quadratic function is calculated and stored as a ridge point.

3. The ridge points are converted into cylindrical, star coordinates as follows:

$$\begin{aligned} \theta^* &= \text{(azimuth for this terrain section)} \\ \rho^* &= \rho \\ z^* &= \rho \tan \beta \end{aligned} \tag{3.1}$$

where ρ and β are the horizontal distance and elevation angle, respectively, from the vantage to each skyline point.

Figure 3.1 shows a terrain section in which two ridge points have been found.

The increments of ρ in the samples of a terrain section are made smallest near the rover, because distortions in the data (caused by sampling and interpolation) have their greatest effect on the skyline image when ρ is small. A quadratic function is used for ρ; the sequence of ρ increments is linear. The quadratic function was chosen over, say, an exponential or reciprocal function, to avoid over-dramatizing the effect. The result yields dense samples near the rover, without unreasonably sparse samples farther out.

3.2 Grouping point into curves

As collected above, the skyline points comprise a big set of individual points. The next step is to group the points into curves, where each curve is represented by an ordered set of points. Points are "connected" into curves according to two criteria:

1. The two points are from adjacent terrain sections in terms of how the ridge points were collected, but not from the same one.

2. The difference between the ρ's of the two points must not exceed a certain portion of their average ρ.

The implementation takes advantage of the fact that the ridge points have been collected monotonically in azimuth.

Note that, even after grouping, the ridge curve is still merely a discrete set of points. The next section discusses how a continuous curve segment is reconstructed from the discrete curve data.

3.3 Interpolating skyline curves

Each skyline is interpolated into a smooth curve using natural cubic splines of ρ and z parameterized by θ. The splines result in C2-continuous functions[3] $\rho(\theta)$ and

[3]C2-continuous means that not only function $\rho(\theta)$ is continuous, but also the first and second derivatives of ρ with respect to θ are continuous. The same is true for $z(\theta)$ and its first and second derivatives.

$z(\theta)$ passing through the individual skyline points collected from either the global map or the sensor data.

The cubic spline interpolation is performed with the `spline()` and `splint()` subroutines of [18]. Interpolation of $\rho(\theta)$ is described below; the procedure for interpolating $z(\theta)$ is identical.

The `spline()` routine calculates a list of second derivatives, $\rho'' = d^2\rho/d\theta^2$, to accompany the discrete θ and ρ values. The spline can then be interpolated (with function `splint()`) as:

$$\rho(\theta) = A\rho_j + B\rho_{j+1} + C\rho_j'' + D\rho_{j+1}'' \tag{3.2}$$

where θ_j, ρ_j, ρ_j'', θ_{j+1}, ρ_{j+1}, and ρ_{j+1}'' are the stored spline parameters at each end of the interval containing θ, and

$$A = \frac{\theta_{j+1} - \theta}{\theta_{j+1} - \theta_j} \tag{3.3}$$
$$B = \frac{\theta - \theta_j}{\theta_{j+1} - \theta_j} \tag{3.4}$$
$$C = \frac{1}{6}(A^3 - A)(\theta_{j+1} - \theta_j)^2 \tag{3.5}$$
$$D = \frac{1}{6}(B^3 - B)(\theta_{j+1} - \theta_j)^2 \tag{3.6}$$

This interpolation has the following properties:

1. Each segment is a polar cubic polynomial, i.e. a cubic polynomial in polar coordinates.

2. The spline passes through all points in the list of discrete samples.

3. The spline is C2-continuous, meaning it is continuous, and has continuous first and second derivatives.

4. The second derivative. ρ'' is constrained to be zero at the two endpoints of the interpolated curve. This property makes it a so-called "natural spline."

The first derivative may be calculated as:

$$\frac{d\rho(\theta)}{d\theta} = \frac{\rho_{j+1} - \rho_j}{\theta_{j+1} - \theta_j} - \frac{3A^2 - 1}{6}(\theta_{j+1} - \theta_j)\rho_j'' + \frac{3B^2 - 1}{6}(\theta_{j+1} - \theta_j)\rho_{j+1}'' \tag{3.7}$$

Once the splines are calculated, the curves are available as continuous functions, $\rho^{\dagger}(\theta^{\dagger})$, $z^{\dagger}(\theta^{\dagger})$ for the dagger curves, and $\rho^{*}(\theta^{*})$, $z^{*}(\theta^{*})$ for the star curves. The interpolated functions are defined throughout the domain were only defined between the endpoints of the curves, and the discrete samples from which the splines were interpolated become transparent to the algorithm.

3.4 Examples of skylines from the global map

Examples of skylines collected from the rough map are shown in Figures 3.2 and 3.3. The 3-D skyline curves are shown in two views— a "forward" view looking out the front of the robot (in spherical dimensions) to show the elevation data, and a "top view" looking down at the curves, to show the curves' ranges from the robot. An additional view shows the robot's location on the terrain for which it collected the skylines.

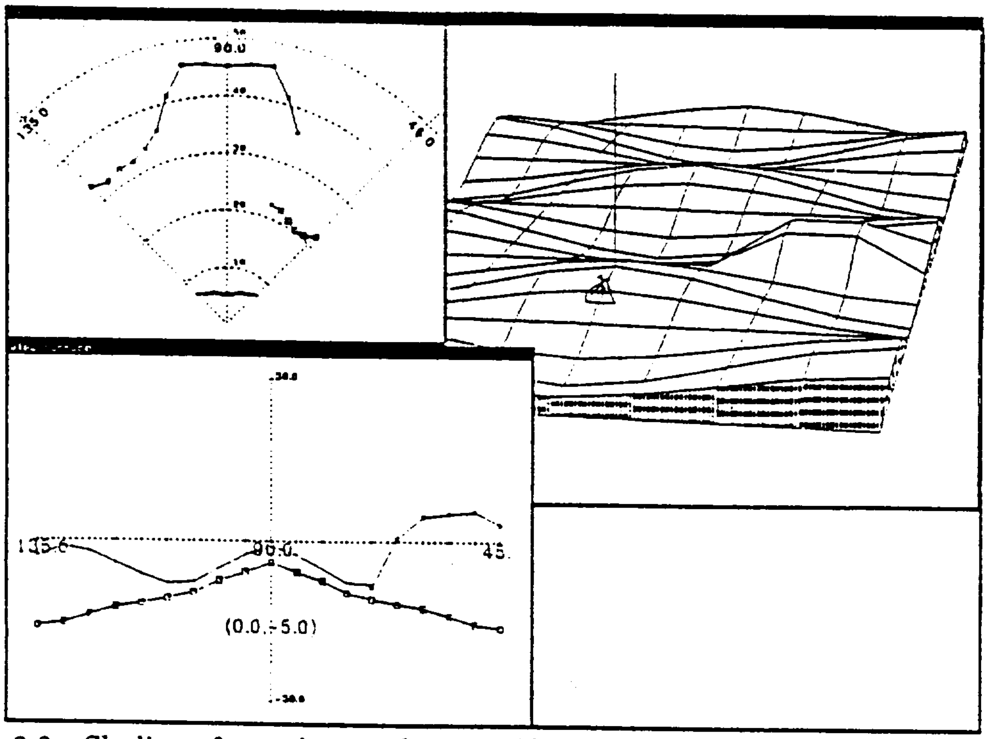

Figure 3.2: Skylines from the rough map: Vantage point (0., -5., 5.28), yaw = 90°.

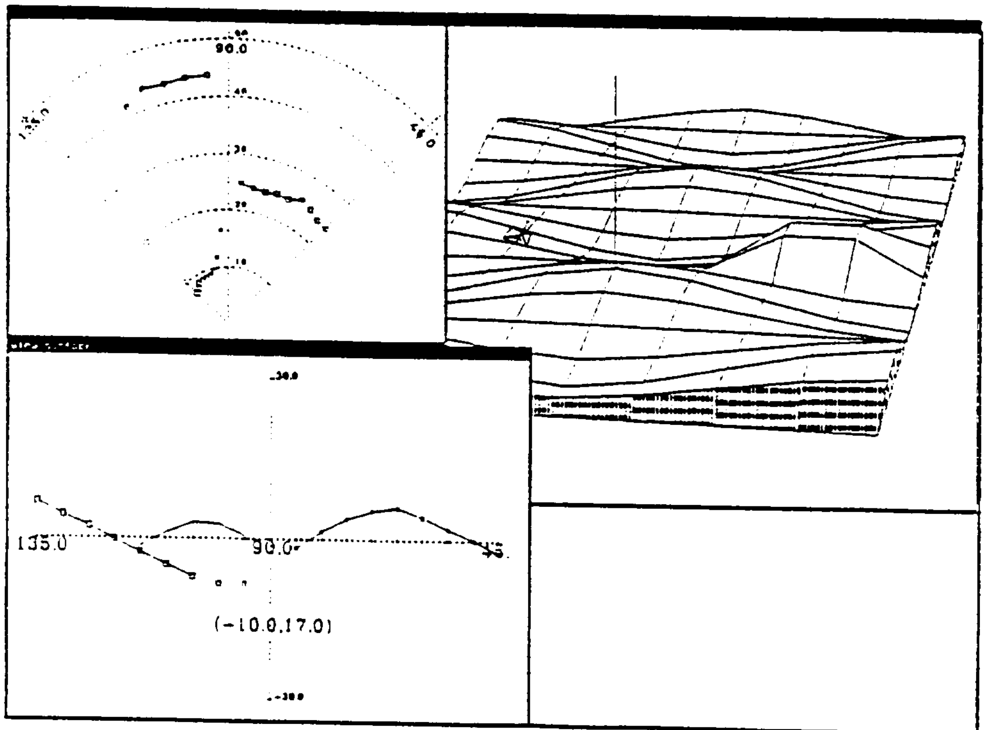

Figure 3.3: Skylines from the rough map: Vantage point (-10., 17., 2.00), yaw = 90°.

CHAPTER 4
TERRAIN MATCHING ALGORITHM

The terrain matching algorithm's task is to find x_v, y_v, and z_v, the translational parameters of the vantage with respect to the global coordinate system (x^g, y^g, z^g), fixed to the terrain. As mentioned previously, the vantage yaw (θ_v) is assumed known, and pitch and roll are assumed to be zero. θ_v is the direction that the vehicle is facing in the horizontal projection, measured counter-clockwise from the x^g-axis.

The terrain matching algorithm receives as input:

1. An initial vantage estimate, based on a previous position of the vehicle, and on the path which it has approximately followed from that position.

2. A set of local skylines, extracted from 3-D vision or laser ranging sensory information. Like the global skylines described in Chapter 3, horizontal radius and vertical height of the local skylines are parameterized as functions of azimuth, but these measurements are of course made with respect to the true or "local" vantage.

It also has access to the global map, from which it can generate a set of global skylines for any trial vantage above the terrain.

To update the vantage estimate, the algorithm first generates a set of "global skylines" from the global map using the procedure described in Chapter 3. It then compares the two sets of skylines in order to relate the local map to the global map and thus determine the vantage. This comparison is formulated in terms of estimating a translation between the set of local skylines and the set of global skylines.

Formulating the terrain matching task in terms of a translation between the curves is described in Section 4.1. The basis of our method for estimating the translation is described in Section 4.2. The overall terrain matching algorithm is "summarized" in Section 4.3, and its computational details are described in Sections 4.4–4.7.

4.1 Formulation: a translation between curves

We repeat Figure 2.1 here as Figure 4.1, this time concentrating on the skylines in the figure. There are three sources of differences between the local (l) and global ($\star$) skylines which will be enumerated shortly; however, we consider the two skylines to represent the *same feature*, from two different vantages. To the extent that the two skylines do represent the same curve with respect to the terrain, their representations with respect to the two vantages are related by a mere translation, and this translation is the error in the current vantage estimate. This is because the vantages themselves are related by a translation (with the orientations equal by assumption). To estimate the translation between the curves with respect to their respective vantages is to estimate the vantage error, which we add to our current vantage estimate for an improved vantage estimate.

Figure 4.1 shows the two skylines with their proper relationships to the global reference. Since we have formulated the problem in terms of a translation between the representations of the skylines with respect to their respective vantages, however, the local skylines are translated (by a yet unknown quantity) so that their vantage is coincident with that of the global ($\star$) skylines.

The translated versions of the local skylines are called the "dagger" ($\dagger$) curves. The relationship of the dagger curves to the dagger system is identical to that of the local curves to the local coordinates, but the dagger coordinate system is coincident with the star system. See Figure 4.2. Of course, no computational conversion from

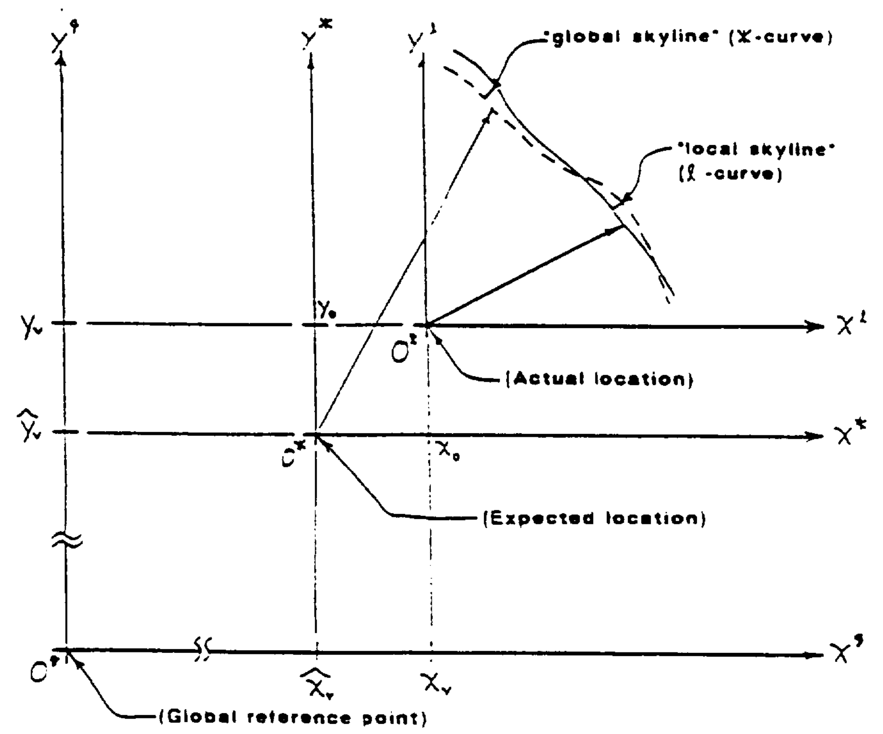

Figure 4.1: Relationships among the global(g), local (l), and star ($\star$) coordinate systems.

local to dagger skylines actually takes place; the translation of the local skylines is merely a distinction in the geometric interpretation of how we are relating the data to each other. The two sets of skylines are thus "overlaid" with each other, in the star and dagger coordinate systems. If the vantage estimate has no error, then the translation of the local skylines to dagger coordinates is null, and the star and dagger curves should remain approximately coincident; if an error is present in the vantage, then the star and dagger curves should differ by a translation equal to the error.

There are three causes of differences between the local and global skylines, even before translating the local curves into dagger curves:

1. Errors in the global map cause the actual (local) and predicted (star) skylines to differ.

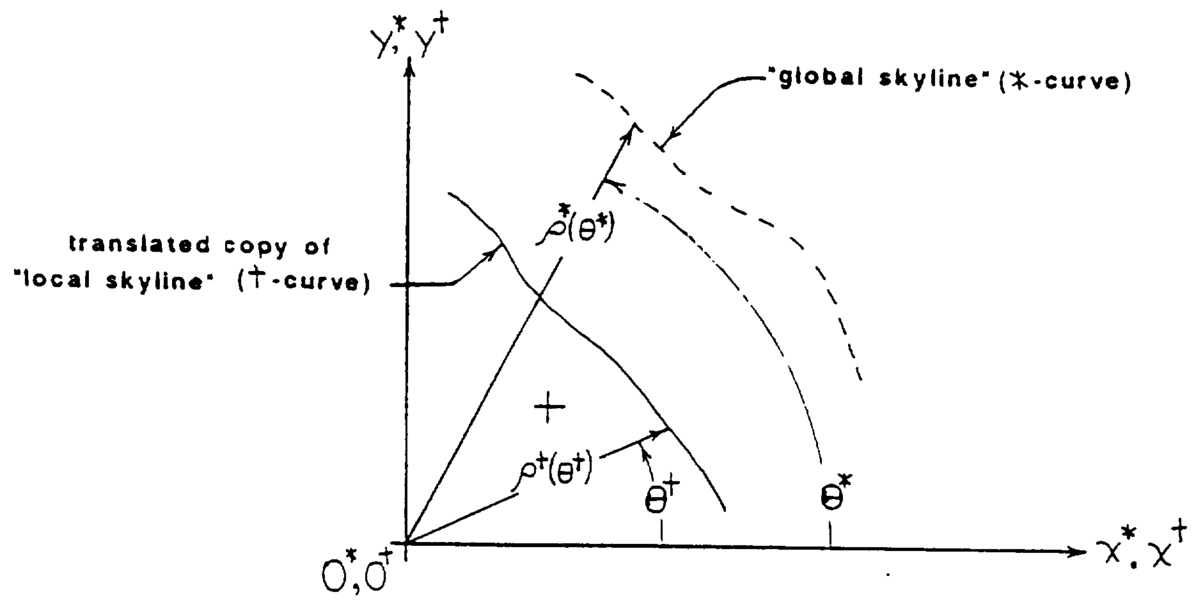
$y^*, y^\dagger$
"global skyline" (*-curve)
translated copy of "local skyline" (†-curve)
$\rho^*(\theta^*)$
$\rho^\dagger(\theta^\dagger)$
$\theta^\dagger$
θ^*
$O^*, O^\dagger$
$x^*, x^\dagger$

Figure 4.2: Illustration of the dagger (†) coordinate system.

2. Any error in the vantage estimate causes differences between the curves, because skylines are vantage-dependent.

3. Numerical errors from sampling and interpolation, both in the generation of global skylines and in the subsequent spline interpolation of both sets of skylines.

As long as the vantage estimate improves in each iteration of the algorithm, the differences resulting from the second cause listed will diminish.

4.2 Method: the Max-Min Principle

We begin the development of our method for finding the translation between two sets of curves by presenting the max-min principle:[4]:

[4]The term "max-min" in this context has nothing to do with max-min game theory in the field of artificial intelligence.

Max-min Principle: *The minimum distance from any point on a curve, to a translated version of the same curve, is less than or equal to the length of the translation.*

(In this case, "same curve" means that it is strictly congruent to the first. Later we will relax this restriction, particularly to allow for a pair of curves whose end points might not correspond, and will have to consider possible exceptions to the original principle.)

The principle is justified as follows: there is at least one point— the corresponding point— on the translated curve which is the length of the translation away from a point on the original curve. There may also be other points on the translated curve which are closer than the corresponding point. Therefore, the minimum distance is less than or equal to the length of the actual translation.

The longest of these minimum-distance vectors is called the *max-min*. We postulate that under most conditions, the max-min approaches the actual translation between the curves. (These conditions, for the max-min computations which we use, will be stated shortly.) The max-min could be used an estimate of the translation itself, but in fact only its direction is used, and the length of the translation in that direction is calculated with an averaging scheme described in Sect. 4.6.

A very direct way to compute the max-min is to compute "shortest-distance" vectors from points on one curve to the other curve, and vice-versa, and to take the longest of these vectors as the max-min. The method we use is similar to this, but not quite the same. The problem is that with piece-wise polar cubic curves (i.e. cubic splines in polar coordinates) a shortest-distance measurement is relatively difficult to compute. Numerically it is expensive, because multiple minima and maxima of distance from the point may exist along the curve, and in polar space the whole curve may have to be searched.

An alternative is to use what we *normal-to-intersect* vectors in place of shortest-distance vectors. A normal-to-intersect vector is a path along the normal line through some point on one curve, to the normal line's intersection with the other curve. (For normal curves from some points, this intersection does not exist, but we will see that this is okay.) The distance along the normal line, in the direction which immediately increases the radius of the line from the origin, is called ϵ; a normal-to-intersect from an "inner curve" (smaller $\rho(\theta)$) to an "outer curve" (larger $\rho(\theta)$) has a positive ϵ, and a normal-to-intersect from an "outer curve" to an "inner curve" has a negative ϵ. The point on the curve, where a normal-to-intersect vector is computed *from* is called its base point. and the point on the other curve which it points *to* is called its terminating point. The curves are also called the base curve and terminating curve, respectively.

Normal-to-intersect vectors are in most cases good substitutes for the shortest-distance vectors. Whereas a shortest-distance vector should be normal to its terminating curve (marking a local minimum in distance), a normal-to-intersect vector is normal to its base curve. By finding the point where a normal line of one curve intersects the other curve, we in most cases obtain a discrepancy vector in the *reverse direction*. (In such cases, the normal-to-intersect vector marks the shortest path from its terminating point to its base curve.) For normal-to-intersects, the search for each ϵ is along a line, and is thus numerically quite straightforward.

Insofar as the normal-to-intersect measurements substitute for discrepancy measurements, the principle above still applies. In addition, the condition under the max-min (as computed with normal-to-intersects) should approach the actual translation is that at least one of the normal-to-intersect vectors points in the direction of the translation between the curves. While we cannot guarantee this ahead of time, we maximize the chances that this occurs by making normal-to-intersect measurements in both directions— from the original curve to the translated curve.

and vice-versa. In short, we propose to use the maximum normal-to-intersect measurement, to estimate the direction of the translation.

4.2.1 Max-min exceptions

This section highlights two kinds of possible exceptions to the max-min principle: exceptions due to non-corresponding endpoints, and exceptions due to using normal-to-intersect measurements instead of discrepancy measurements.

When matching skylines, we must be prepared for the endpoints of the curves to not correspond to each other. That is, while the curves must share some common, corresponding portion of a skyline, one curve may be "longer" than the other on either or both ends. (If the endpoints were known to correspond to each other, then the translation could be easily calculated from endpoints only.) In fact, the endpoints are the most sensitive parts of the skylines (often located at visual inflection points), and will most often *not* correspond to each other due to data noise and vantage error.

If discrepancies themselves were being measured (as opposed to normal-to-intersects), then discrepancies near noncorresponding endpoints could be longer than the translation between the curves, and have little to do with the direction or translation between the curves; this could disrupt the entire algorithm. A way to avoid this case would be to discard any discrepancy whose terminating point is an endpoint of its terminating curve. A practically equivalent condition would require the discrepancy vector to be normal to the terminating curve at its terminating point, but merely comparing it to the endpoints is easier.

Even following this rule still allows for cases, for example see Figure 4.3, where noncorresponding endpoints can cause extra-long discrepancies. Such cases, however, are expected to be extremely rare, and have not been further addressed.

A different type of exception is made possible by using normal-to-intersect

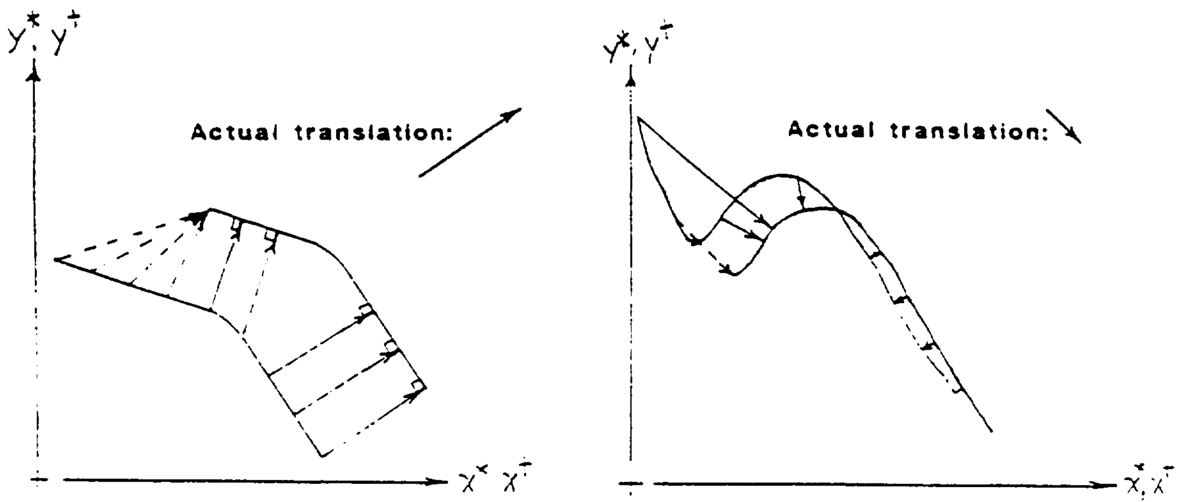

Figure 4.3: Max-min exceptions due to noncorresponding endpoints: on the left, discarding the discrepancies which terminate on endpoints is sufficient, but on the right it is not. (Dashed vectors represent discarded discrepancies.)

measurements in place of discrepancy measurements. This exception, illustrated in Figure 4.4, may result from a noisy base curve, or from a pair of curves with a distinct irregularity.

The spline interpolation in Sect. 3.3 is typically smooth (C-2 continuous), and filtering techniques may be applied in the future as necessary in order to prevent noise from causing this type of exception. The case of a distinct irregularity— referring in the right-hand side of Figure 4.4 to the "bump" common to both curves— may demand more attention in the future. To a human matching the curves intuitively, an irregularity common to both curves provides valuable clues; an autonomous matching scheme should be able to at least tolerate, if not thrive upon, such potentially valuable information. (The discrepancy measurements do in fact take advantage of such an irregularity, because the peak in discrepancy length, taken in a certain direction, is very sharp.)

One way to handle this exception may be to make sure that the distance from a normal-to-intersect's terminating point to its base curve is in fact a local

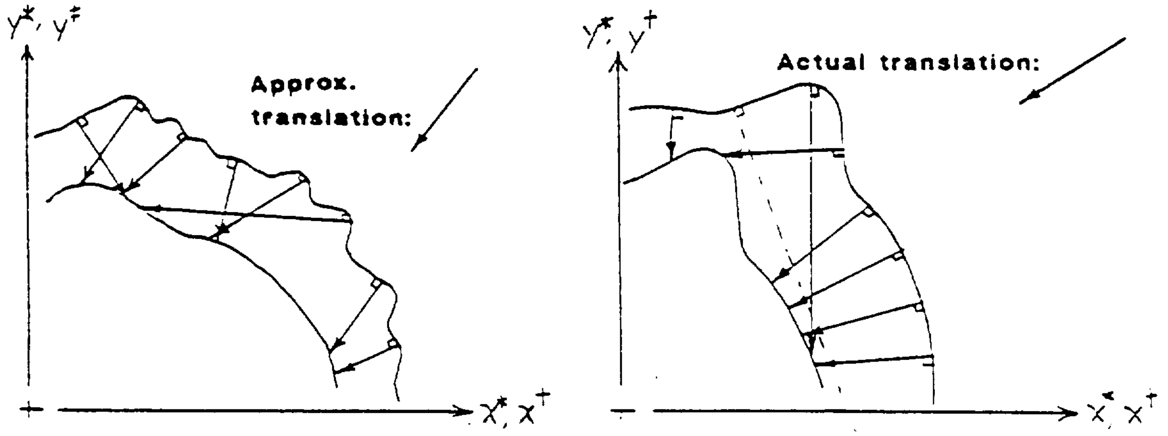

Figure 4.4: Max-min exceptions due to using normal-to-intercepts on noisy or sharply turning data.

minimum at its base point compared to its neighbors on the base curve, and not a local maximum. Merely the fact that it is normal to its base curve allows for either a maximum, a minimum, or an inflection, but to take the place of a "reverse discrepancy" it must be a minimum. A test for this compares the curvature of the base curve at the base point, to the measured ϵ, but this test was not implemented for the experiments described in this paper.

4.3 Implementation: terrain matching algorithm

The initial vantage estimate is expressed as $(\hat{x}_v{}^0, \hat{y}_v{}^0, \hat{z}_v{}^0)$, and the vantage estimate after i iterations of the algorithm is $(\hat{x}_v{}^i, \hat{y}_v{}^i, \hat{z}_v{}^i)$. The error in the vantage estimate after i iterations is denoted (x_0^i, z_0^i, z_0^i):

$$\text{For } i \geq 0,\quad \begin{aligned} x_v &= \hat{x}_v{}^i + x_0^i \\ y_v &= \hat{y}_v{}^i + y_0^i \\ z_v &= \hat{z}_v{}^i + z_0^i \end{aligned} \tag{4.1}$$

The ith iteration of the algorithm consists of the following steps:

1. Generate the global skylines which, according to the rough map, are the skylines visible from the most recent vantage estimate $(\hat{x_v}^{i-1}, \hat{y_v}^{i-1}, \hat{z_v}^{i-1})$.

2. Perform the "curve correspondence," determining which global skylines correspond to which local skylines. Its output is a list of corresponding curve pairs, each pair consisting of one global and one local skyline which represent roughly the same skyline from the two sources of information.

3. Compute the "max-min," which is the longest (max) of the minimum-distance vectors between corresponding pairs of local and global skylines, although the minimum-distance vectors are approximated in the reverse direction by a "normal-to-intersect" procedure.

4. Estimate the vantage error $(\hat{x_0}^{i-1}, \hat{y_0}^{i-1}, \hat{z_0}^{i-1})$, by using the max-min's direction as a rule for point-by-point correspondence and averaging the distances between the curves taken in that direction.

5. Update the vantage estimate by the estimate of its error:
$$\begin{aligned}\hat{x_v}^{i} &= \hat{x_v}^{i-1} + \hat{x_0}^{i-1}\\ \hat{y_v}^{i} &= \hat{y_v}^{i-1} + \hat{y_0}^{i-1}\\ \hat{z_v}^{i} &= \hat{z_v}^{i-1} + \hat{z_0}^{i-1}\end{aligned} \qquad (4.2)$$
(Compare to Equ. 4.1.)

The algorithm repeats this cycle until the magnitude of the difference between two successive vantage estimates, that is of the vantage error estimate, is less than a certain threshold. The details of the steps listed above are, except for the curve correspondence, given in the following sections. The curve correspondence algorithm has not yet been implemented and has so far been performed interactively (by a person) rather than automatically.

From here forward we will drop the i superscripts; by default we will be in the $(i+1)$th iteration, calculating quantities with the i superscript.

4.4 Curve correspondence

After both sets of skylines are obtained, the curve correspondence must be performed, to associate curves from one set with specific curves in the other. As will be shown in the Results section, multiple correspondences where more than one local skyline corresponds to the same global skyline or vice-versa are possible.

In the following sections, references to the sets of skylines are implicitly limited to those skylines which were found to have corresponding curves in the other set.

4.5 Computing the max-min

Computing the max-min means selecting the longest normal-to-intersect in either direction, from the dagger curve to the star curve or vice-versa.

We now describe the normal-to-intersect procedure, for intersecting the normal to a point on the dagger curve with the star curve. Computing a normal-to-intersect in the reverse direction is identical except for swapping the curves.

The first step is to find an expression for the line which is normal to the dagger curve at some $\theta = \theta_n$ (n subscript is for "normal point"). Define ϕ as the angle that the dagger curve at $(\theta_n, \rho^\dagger(\theta_n))$ makes with a curve of constant ρ through that point.

$$\phi = \arctan\left(\frac{d\rho^\dagger(\theta_n)}{\rho^\dagger(\theta_n)d\theta}\right) \tag{4.3}$$

(ϕ is always between -90° and 90°.) The line tangent to the curve points in the direction of $(\theta_n - \phi + 90°)$, and the normal line points in the direction of $(\theta_n - \phi)$. Next, parameterize the normal line in terms of ϵ, which is the distance along the line in the normal direction. Letting the l subscript denote "line," the normal line is parameterized in cylindrical coordinates about the dagger origin as:

$$\rho_l(\epsilon) = \sqrt{(x_n + c_x\epsilon)^2 + (y_n + c_y\epsilon)^2} \tag{4.4}$$

$$\theta_l(\epsilon) = \text{atan2}\left((x_n + c_x\epsilon), (y_n + c_y\epsilon)\right) \tag{4.5}$$

where

$$\begin{aligned} x_n &= \rho^{\dagger}(\theta_n)\cos\theta_n \\ y_n &= \rho^{\dagger}(\theta_n)\sin\theta_n \end{aligned} \tag{4.6}$$

and

$$\begin{aligned} c_x &= \cos(\theta_n - \phi) \\ c_y &= \sin(\theta_n - \phi) \end{aligned} \tag{4.7}$$

At the point where the normal line intersects the star curve, $\rho_l(\epsilon) = \rho^{\star}(\theta_l(\epsilon))$. Thus, define

$$f(\epsilon) = \rho^{\star}(\theta_l(\epsilon)) - \rho_l(\epsilon) \tag{4.8}$$

and our task is to search for $\epsilon : f(\epsilon) = 0$. This search is performed numerically using the `zbrac()` and `zbrent()` subroutines of [18].

The `zbrac()` routine brackets a root, meaning it attempts to find an ϵ_1 and ϵ_2 such that $f(\epsilon_1)$ and $f(\epsilon_2)$ have opposite signs. Under this condition, a root is guaranteed to exist between ϵ_1 and ϵ_2. It is initialized with the interval from 0 to $f(0) = \rho^{\star}(\theta_n) - \rho^{\dagger}(\theta_n)$, and expands if necessary until it brackets a root. If it fails to bracket a root after a specified number of expansions, then the normal-to-intersect is considered not to exist for this θ_n. The `zbrent()` routine contracts the interval ϵ_1, ϵ_2 surrounding the root until it is sufficiently small to provide the required precision. Additional details of `zbrac()` and `zbrent()` are well documented in [18].

The subroutine which calculates $f(\epsilon)$ produces a value for $\rho^{\star}(\theta_l(\epsilon))$ even when $\theta_l(\epsilon)$ is outside the star curve's valid range, by extrapolating the cubic equation of the nearest cubic piece of the star curve to the invalid $\theta_l(\epsilon)$. The algorithm checks that the solution makes a valid intersection with the star curve, *after* `zbrent()` returns a root. If the resulting $\theta_l(\epsilon)$ lies outside the valid bounds of the star curve, then this means that the normal line does not intersect the corresponding curve at all, and the normal-to-intersect is considered not to exist for this θ_n, just as if a solution had not been bracketed.

Normal-to-intersect calculations are made from evenly spaced points on all curves, to their corresponding curves. The normal-to-intersect vectors are calculated in both directions— from dagger to star and vice-versa— because the normal-to-intersect distances sometimes exhibit distinct peaks only in one of these directions.

The normal-to-intersect vector with the overall biggest magnitude (absolute value) is selected as the max-min; it is the longest normal-to-intersect vector. The max-min is represented by its ϵ, and its c_x and c_y of Equ. 4.7 which represent the direction of the normal-to-intersect vector. The direction, in terms of whether it is a dagger-to-star normal-to-intersect or a star-to-dagger curve normal-to-intersect, is also noted.

4.6 Direction-to-intersect measurements

This section describes how the vantage error (x_0, y_0, z_0) is estimated, once the max-min is computed. To repeat, the vantage error is assumed to be a translation between from the dagger to the star skylines.

The direction of the max-min is effectively used as a rule of point-by-point correspondence between the two sets of curves. That is, we retain the direction of the max-min but not its length, and the length of the translation is then estimated as the *average* of the distance between the curves, taken in this direction. A distance from a point on one curve to its corresponding curve, taken in a particular direction, is called a *direction-to-intersect* measurement.

The equations for the horizontal translation are:

$$\hat{x}_0 = c_x \frac{1}{\left(N_{\dagger\star} + N_{\star\dagger}\right)} \left(\sum_{i=1}^{N_{\dagger\star}} \Delta_{\dagger\star i} - \sum_{i=1}^{N_{\star\dagger}} \Delta_{\star\dagger i} \right) \tag{4.9}$$

$$\hat{y}_0 = c_y \frac{1}{\left(N_{\dagger\star} + N_{\star\dagger}\right)} \left(\sum_{i=1}^{N_{\dagger\star}} \Delta_{\dagger\star i} - \sum_{i=1}^{N_{\star\dagger}} \Delta_{\star\dagger i} \right) \tag{4.10}$$

where each Δ_i is a direction-to-intersect measurement from one curve to another:

$N_{\dagger\star}$ and $N_{\star\dagger}$ are the number of direction-to-intersects which intersected in the respective directions. The star-to-dagger measurements are negated for purposes of calculating $\hat{x}_0$ and $\hat{y}_0$. Computationally, the direction-to-intersects are equivalent to normal-to-intersects, but their directions represented by c_x and c_y are the same as the max-min's, rather than calculated with Equ. 4.7 for each measurement.

Direction-to-intersects are calculated from the dagger curves at equal increments of $\theta^\dagger$, and from the star curves. However, only the direction-to-intersects that do intersect the other curves are included in $N_{\dagger\star}$ and $N_{\star\dagger}$, and are used in Eqs. 4.9 and 4.10. The averaging acts to filter out errors in the max-min due to noise in the curve data.

4.7 Vertical translation

Once the direction-to-intersects are computed, the vertical distances are also readily available; the z values at the base and terminating points of each direction-to-intersect vector can be computed from the $\theta^\dagger$ and $\theta^\star$ values at these points. $\hat{z}_0$ is computed as the average difference between these heights:

$$\hat{z}_0 = \frac{1}{N_\Delta} \sum_{i=1}^{N_\Delta} \left(z^\star(\theta^\star_{\Delta i}) - z^\dagger(\theta^\dagger_{\Delta i}) \right) \tag{4.11}$$

In this equation, N_Δ equals $N_{\dagger\star} + N_{\star\dagger}$ of Eqs. 4.9 and 4.10; the summation is over all connected direction-to-intersects.

The $\theta^\dagger_\Delta$ and $\theta^\star_\Delta$ denote the θ values at the base and terminating points of a direction-to-intersect vector (not necessarily respectively). These values are readily available during the direction-to-intersect computations for Eqs. 4.9 and 4.10. Consider for instance a dagger-to-star direction-to-intersect: $\theta^\dagger_\Delta$ is pre-specified, and $\theta^\star_\Delta$ is obtained as (apply Equ. 4.5, replacing the normal-to-intersect distance ϵ with the direction-to-intersect distance Δ):

$$\theta^\star_\Delta = \theta_l(\Delta) = \operatorname{atan2}\left((x_n + c_x\Delta), (y_n + c_y\Delta)\right) \tag{4.12}$$

where

$$\begin{aligned} x_n &= \rho^\dagger(\theta_\Delta^\dagger)\cos\theta_\Delta^\dagger \\ y_n &= \rho^\dagger(\theta_\Delta^\dagger)\sin\theta_\Delta^\dagger \end{aligned} \tag{4.13}$$

c_x and c_y represent the direction of the max-min; and Δ is the direction-to-intersect distance. Eqs. 4.9, 4.10, and 4.11 are thus calculated simultaneously with each other, one direction-to-intersect at a time.

The vantage error estimate $(\hat{x_0}, \hat{y_0}, \hat{z_0})$ thus calculated is added to the previous vantage estimate in each iteration, to produce a revised vantage estimate.

CHAPTER 5
SIMULATION RESULTS

The terrain and map modelling, skyline generation, and terrain matching algorithms of Chapters 2, 3, and 4, respectively were simulated in the form of C programs on Sun 3 workstations. This chapter describes the main results of our simulations. A brief description of the software is provided in Appendix A.

5.1 Individual pairs of curves

This section presents the results of applying the curve-matching technique to a single pair of test curves, under two conditions: one with noise in the data, and one without. We demonstrate the algorithm on a single pair of curves in order to show the individual steps of the terrain matching algorithm. When full sets of skylines are compared to each other, the large amount of intermediate data involved is difficult to convey in a meaningful way.

Each experiment performed one iteration of the following sequence of steps:

- Compute the normal-to-intersect vectors.
- Form epsilon profiles of the normal-to-intersect measurements, to determine the max-min.
- Compute the direction-to-intersect vectors, in the direction of the max-min.
- Average the direction-to-intersect measurements to estimate the horizontal translation between the curves.
- Find the vertical translation, using the max-min as a rule of point-by-point correspondence between the curves.

The results were generated by program **mm4**, (see Appendix A).

The test curves used here were generated from an analytical function (an exponential spiral) in cylindrical coordinates. Sample points of the analytical function were taken for each of two curves in the test. The two sets of samples were taken over different (but largely overlapping) segments of the spiral, so that the endpoints of the test curves did not correspond directly to each other. Then one of the sets of sample points was converted to Cartesian coordinates, translated, and re-converted into cylindrical coordinates. Finally, each curve was interpolated with natural cubic splines in the cylindrical coordinates. In the case of the "noisy" experiment, each sample point was perturbed with independent, zero-mean Gaussian-distributed errors in the ρ and z components, prior to cubic spline interpolation. Different standard deviations were specified for the ρ and z errors.

We interpret the two curves, shown in Figures 5.1 and 5.2, as the "star" and "dagger" curves in the terminology of Chapter 4. They are assumed to represent two perceptions of the same curve from two different vantages (but with non-corresponding endpoints). The star curve represents the global skyline for an "incorrect" vantage estimate, and the dagger curve represents a local skyline, conceptually translated from its unknown but true position, to the known but incorrect vantage estimate. The translation which the local skyline underwent was actually the negative of the error in the vantage estimate $(\tilde{x}_0, \tilde{y}_0, \tilde{z}_0)$, and the algorithm proceeded to estimate this translation by "matching" the skylines back together.

Our experiments used the following analytical functions for generating the test curves:

$$\rho_{sp}(\theta) = 10.0e^{.4\theta} + 1.0 \tag{5.1}$$

$$z_{sp}(\theta) = 3.0\cos\left[1.5(\theta - \frac{\pi}{4})\right] \tag{5.2}$$

The dagger curve consisted of 10 samples, evenly spaced in θ from 10° to 100°. The star curve consisted of 10 samples in the range from −25° to 90°, and the samples

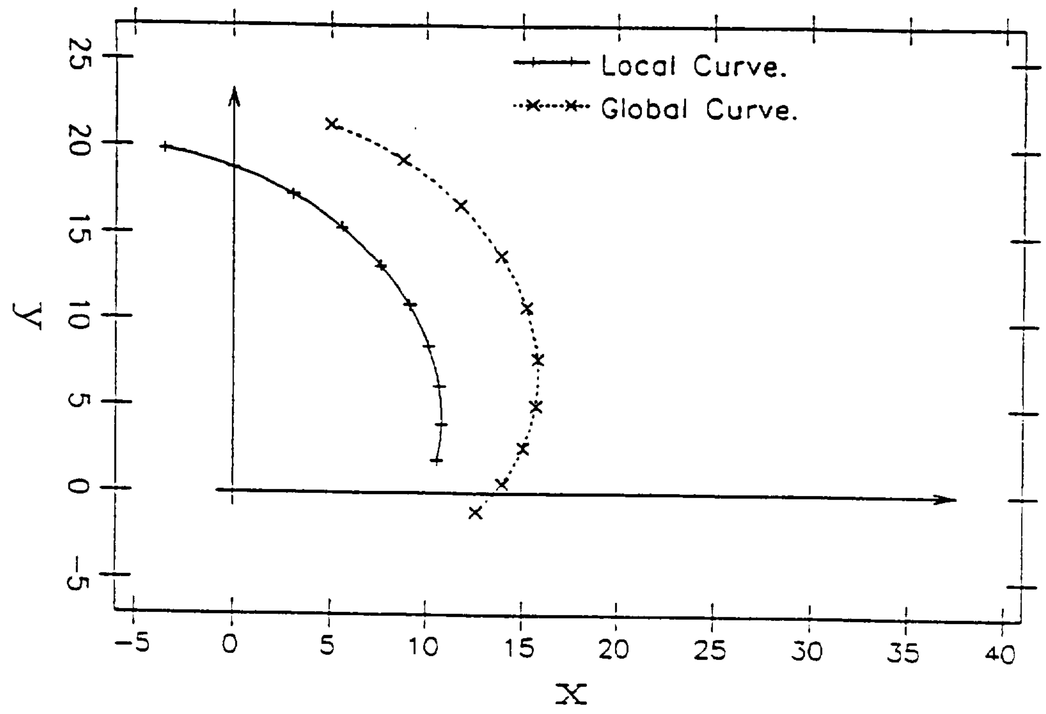

Figure 5.1: Star and dagger curves for noiseless test case.

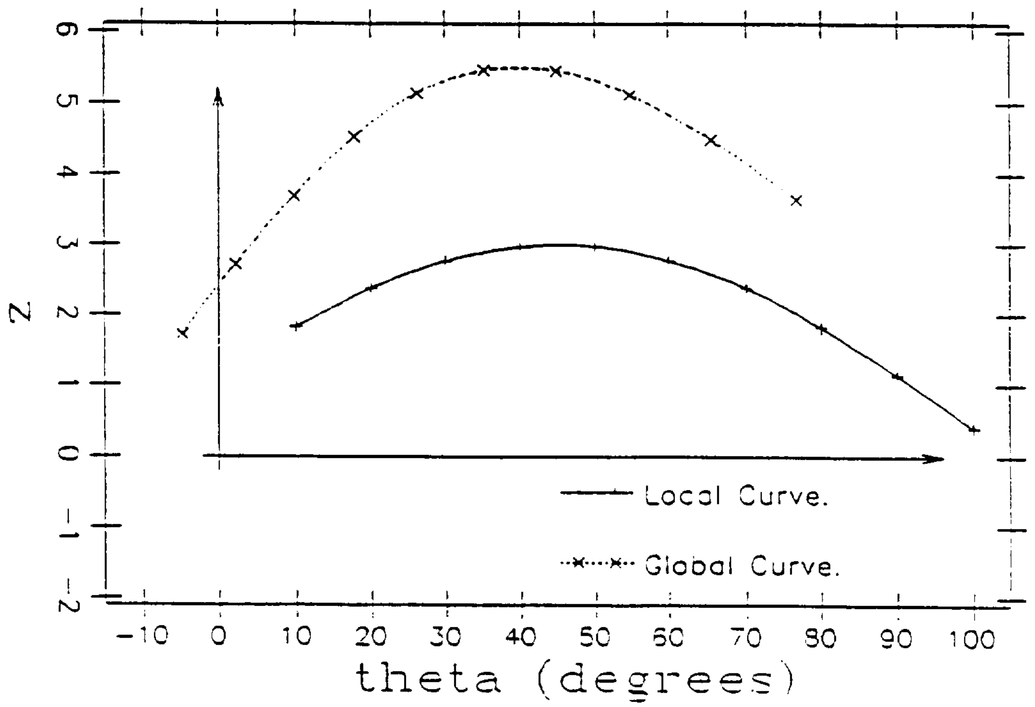

Figure 5.2: Height data for the noiseless star and dagger curves.

were then translated by $(x_o, y_o, z_o) = (5., 2.5, 2.5)$. The formulas for carrying out this translation are:

$$\theta_i^* = \arctan\left(\frac{\rho_{sp}(\theta_i)\sin\theta_i + y_0}{\rho_{sp}(\theta_i)\cos\theta_i + x_0}\right) \tag{5.3}$$

$$\rho_i^* = \sqrt{(\rho_{sp}(\theta_i)\sin\theta_i + y_0)^2 + (\rho_{sp}(\theta_i)\cos\theta_i + x_0)^2} \tag{5.4}$$

$$z_i^* = z_{sp}(\theta_i) + z_0 \tag{5.5}$$

After the translation, θ^* ranged from about 1.1° to 76.8°. In Figure 5.1, the samples from which the curves were interpolated are highlighted by **x**'s and +'s.

The results of the noiseless experiment, which matched the curves shown in Figure 5.1, are shown in Figures 5.3–5.9. Figures 5.3 and 5.4 show the normal-to-intersect vectors. As explained in Section 4.2, the normal-to-intersect measurements were made in both directions. The longest of all of these was chosen as the maxmin. Next are the direction-to-intersect vectors, Figures 5.5 and 5.6, computed in the direction of the max-min.

The translation is estimated as the average of the direction-to-intersect vectors. In this case, the estimated translation was (5.040, 2.425, 2.500), compared to a "true" value of (5., 2.5, 2.5). Figure 5.7 shows that the curves are indeed well matched when the local curve is translated back according to the estimated translation between the curves. In this figure we denote the re-translated curves as "double-dagger" (‡) curves. Whereas the origin of the star and dagger systems is the estimated vantage prior to processing, the double-dagger origin is the estimate after processing. This third coordinate system for the local curves is illustrated in Figure 5.8. In a successful match, the double-dagger curves are roughly coincident with the star curves.

The matching the vertical data is illustrated in Figure 5.9. This figure deserves some explanation: the star curve is still plotted as $z^*(\theta^*)$ vs. θ^*, as in Figure 5.2. The double-dagger z function, however, is also plotted as a function of θ^*, because

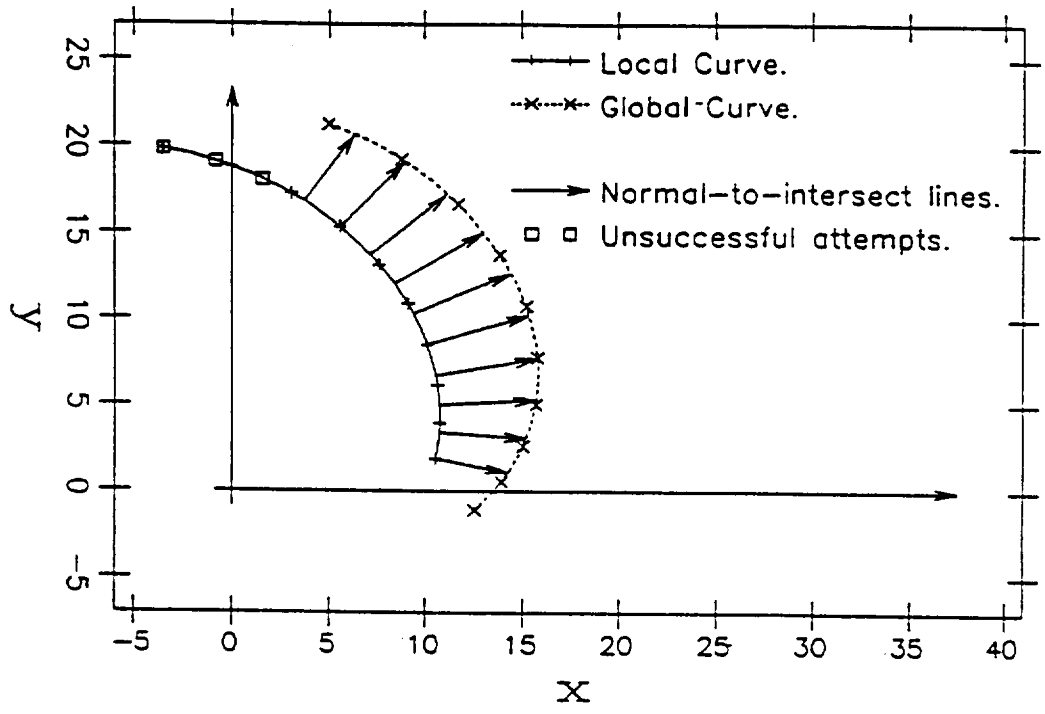

Figure 5.3: Normal-to-intersect vectors from the noiseless dagger curve to the star curve.

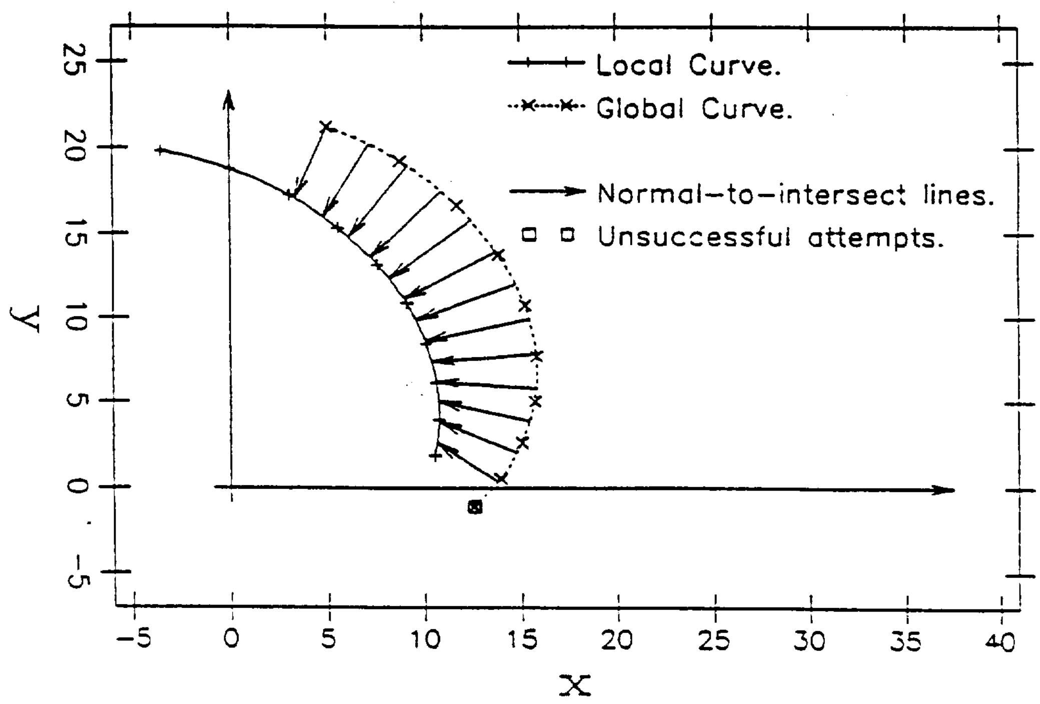

Figure 5.4: Normal-to-intersect vectors from the noiseless star curve to the dagger curve.

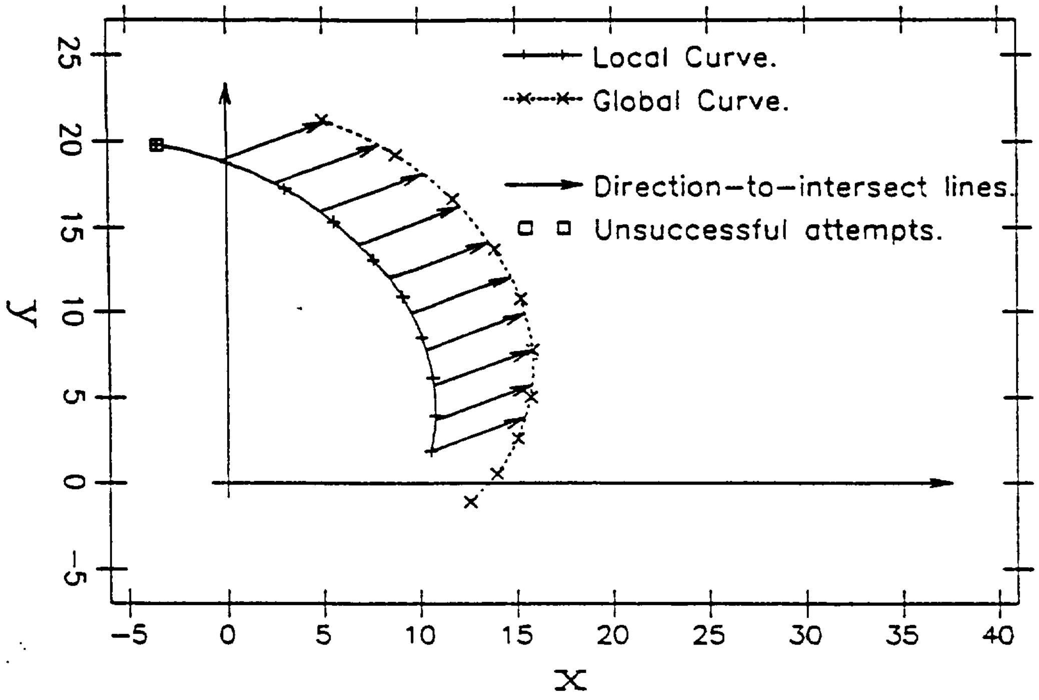

Figure 5.5: Direction-to-intersect vectors from the noiseless dagger curve to the star curve.

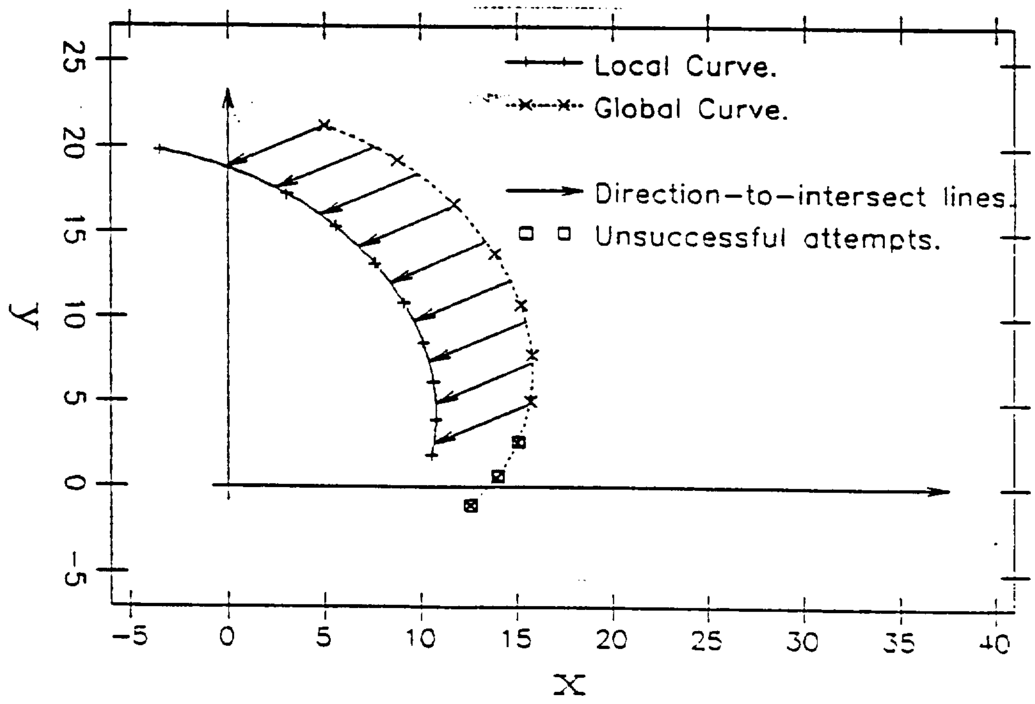

Figure 5.6: Direction-to-intersect vectors from the noiseless star curve to the dagger curve.

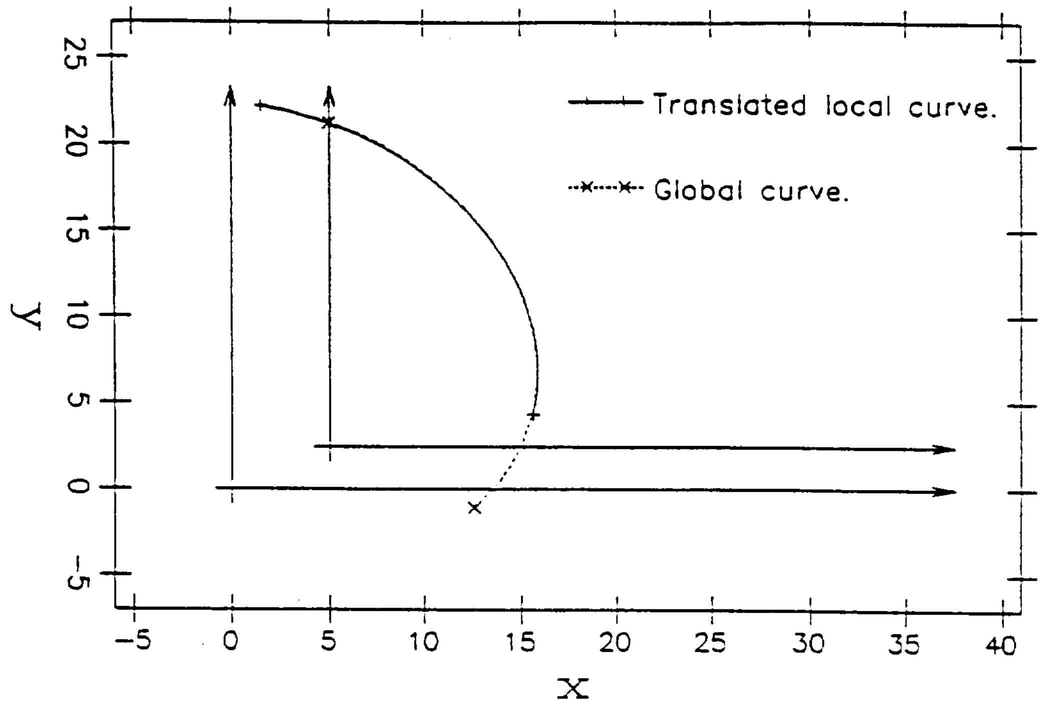

Figure 5.7: Curve match based on direction-to-intersect averaging for the noiseless curves.

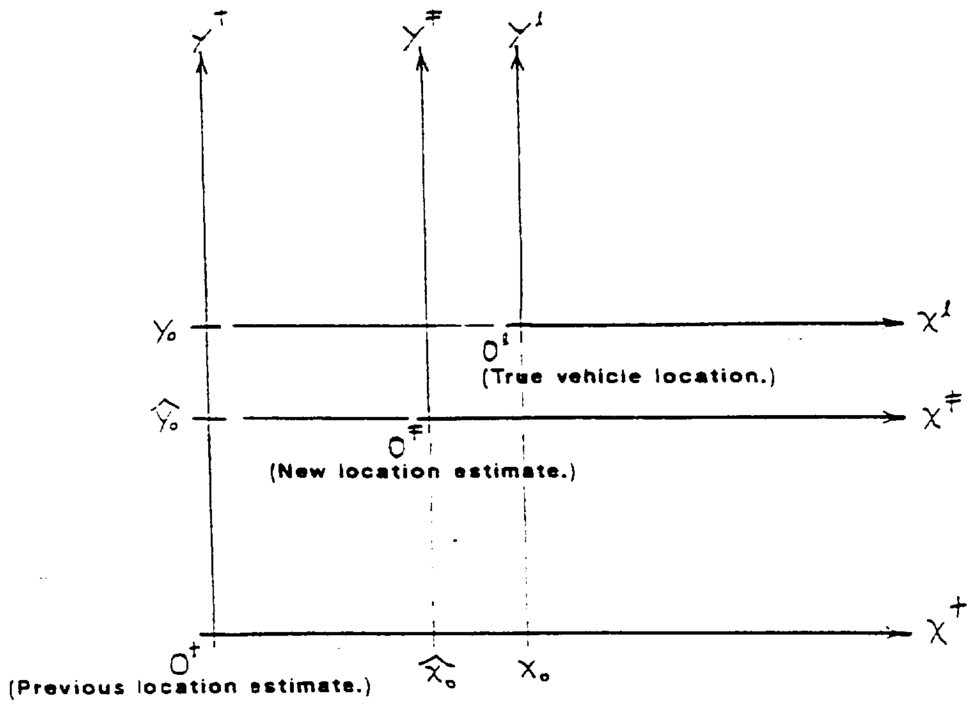

Figure 5.8: Three versions of the "local" system: the local (l), dagger (†), and double-dagger (‡) coordinate systems.

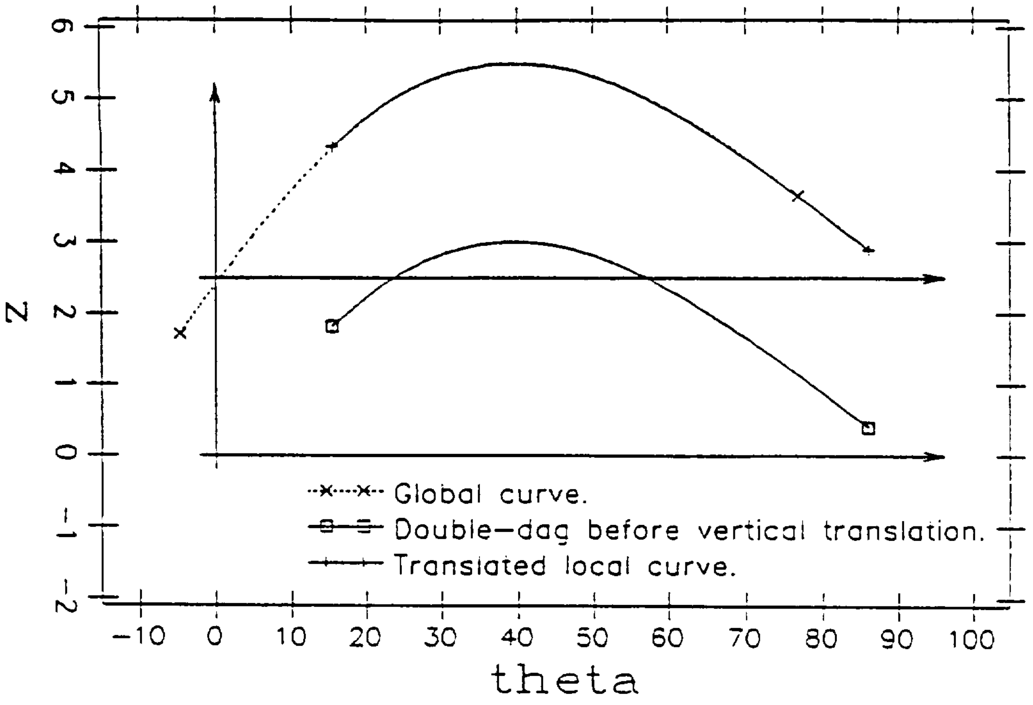

Figure 5.9: Matching of the vertical data for the noiseless case, based on the correspondence developed from the horizontal data.

we want to compare the curves according to the point-by-point correspondence rule that the max-min provides. This requires a transformation relating the two systems, specifically $\theta^{\ddagger}$ to θ^{*}. We call this the correspondence relationship, denoted with a c subscript, and the horizontal geometry of Figure 5.7 indicates that:

$$\theta_c^{\ddagger}(\theta^{*}) = \operatorname{atan2}\left((\rho^{*}(\theta^{*})\sin\theta^{*} - \hat{y}_0), (\rho^{*}(\theta^{*})\cos\theta^{*} - \hat{x}_0)\right) \qquad (5.6)$$

The double-dagger curve in Figure 5.9 is thus $z^{\ddagger}(\theta_c^{\ddagger}(\theta^{*}))$ vs. θ^{*}. The "double-dagger curve before vertical translation" shows the vertical distance between the curves when the horizontal correspondence is applied to the data without computing $\hat{z}_0$. Equ. 4.9 averaged several measurements of this vertical distance to calculate $\hat{z}_0$, which resulted in the double-dagger curve matching the star curve vertically as well as horizontally.

In the noisy version of the experiment, the standard deviation of the errors in ρ and z were .25 and .05 meters, respectively; the resulting noisy curves are

	x_0	y_0	z_0
Actual translation	5.0	2.5	2.5
Estimation without noise	5.040	2.425	2.500
Estimation with noise	4.937	2.770	2.511

Table 5.1: Summary of estimation results for a single pair of curves.

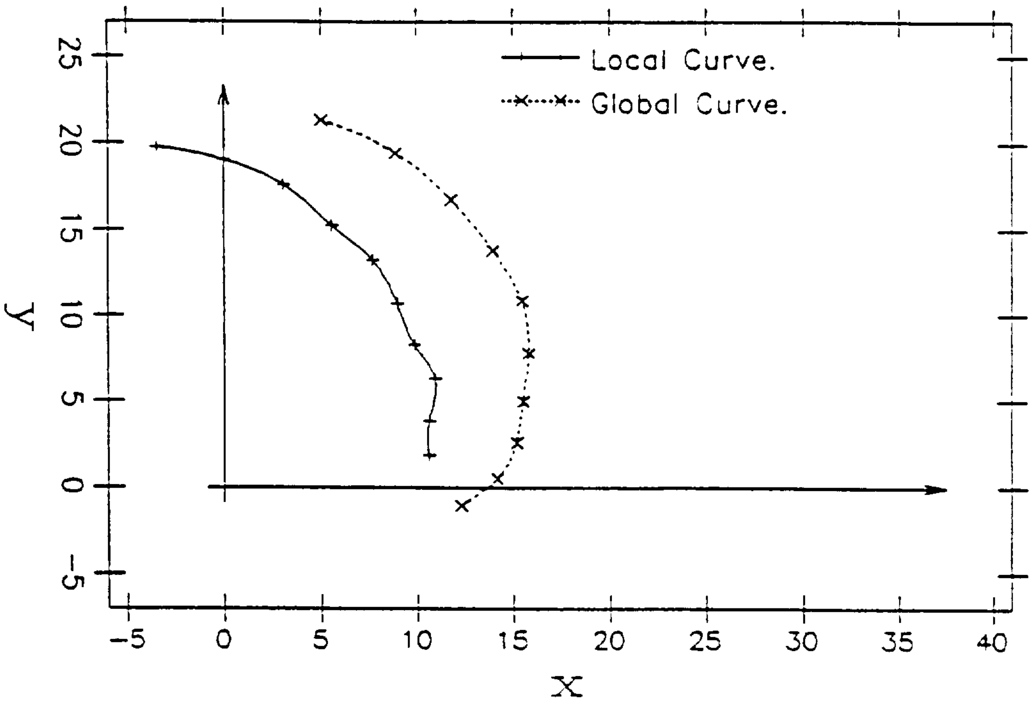

Figure 5.10: Star and dagger curves for noisy test case.

shown in Figures 5.10 and 5.11. Figures 5.12–5.17 illustrate the results of the noisy experiment, in the same format as Figures 5.3–5.7 and 5.9. The results of the both experiments are summarized in Table 5.1.

5.2 Whole sets of curves

This section describes the results of simulating the whole terrain matching algorithm (except curve correspondence). The results were generated with program cm6 (see Appendix A).

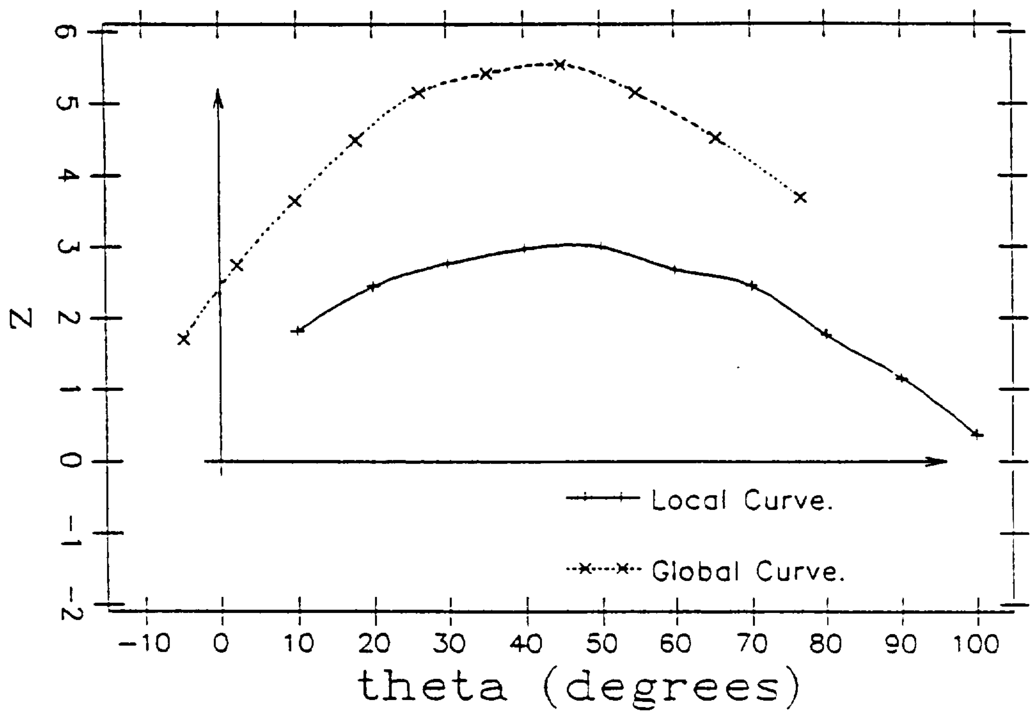

Figure 5.11: Height data for the noisy star and dagger curves.

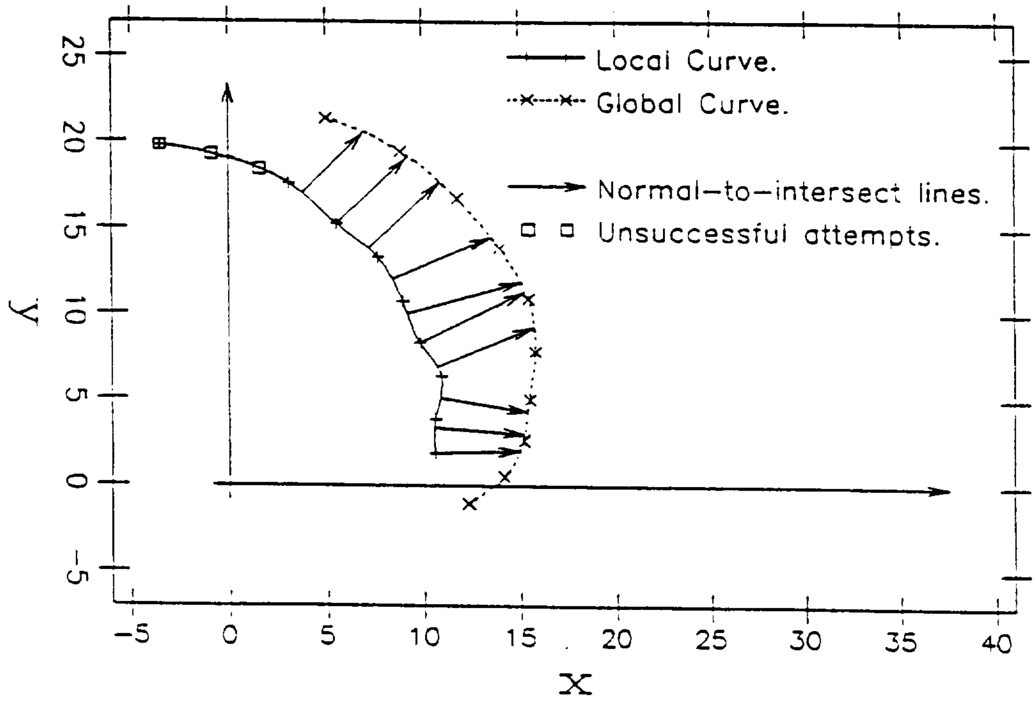

Figure 5.12: Normal-to-intersect vectors from the noisy dagger curve to the star curve.

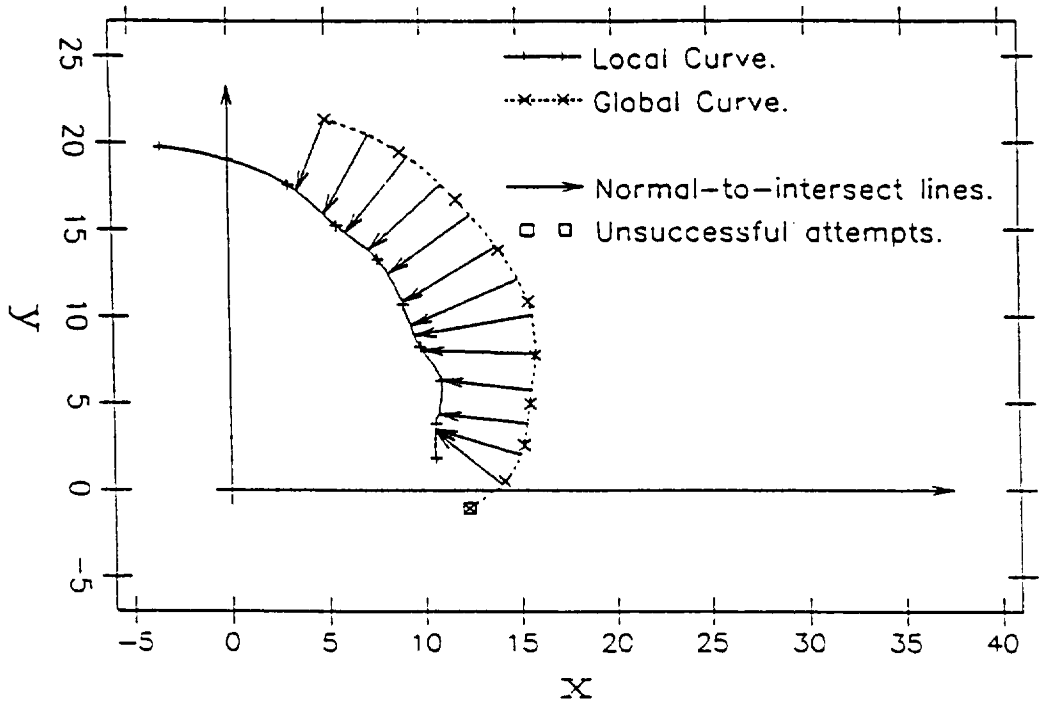

Figure 5.13: Normal-to-intersect vectors from the noisy star curve to the dagger curve.

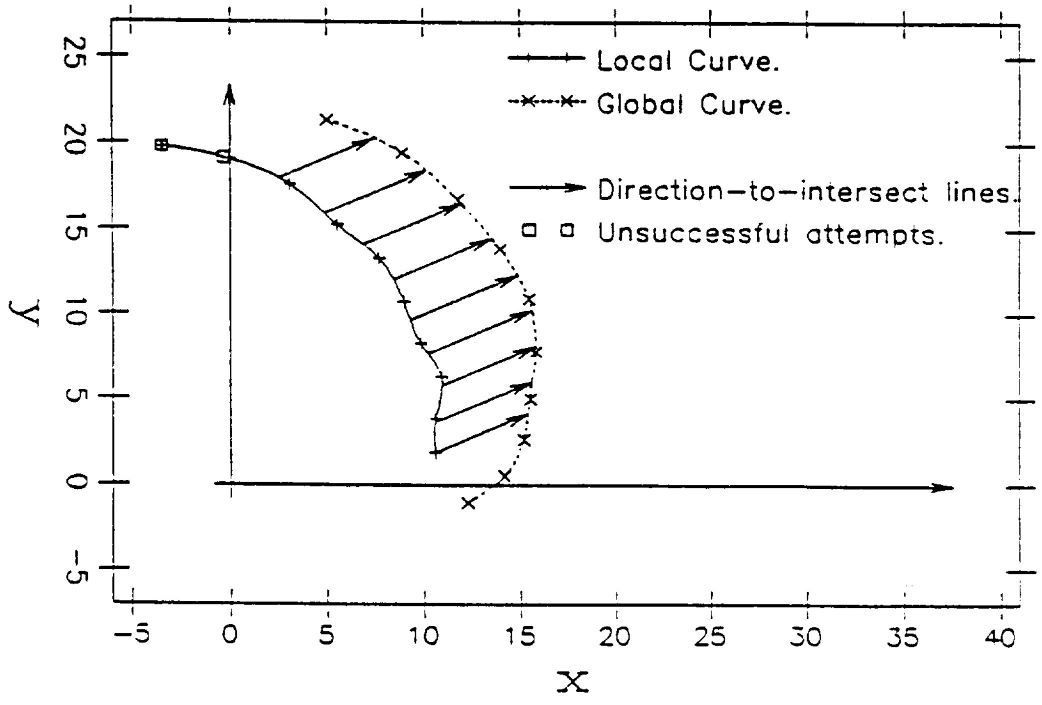

Figure 5.14: Direction-to-intersect vectors from the noisy dagger curve to the star curve.

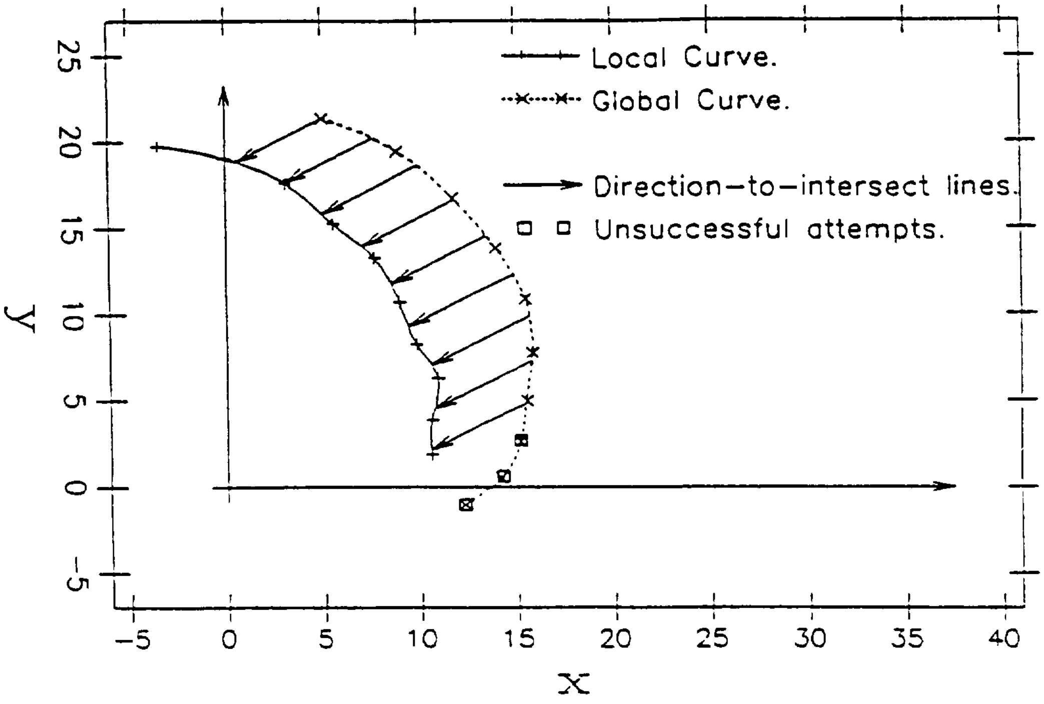

Figure 5.15: Direction-to-intersect vectors from the noisy star curve to the dagger curve.

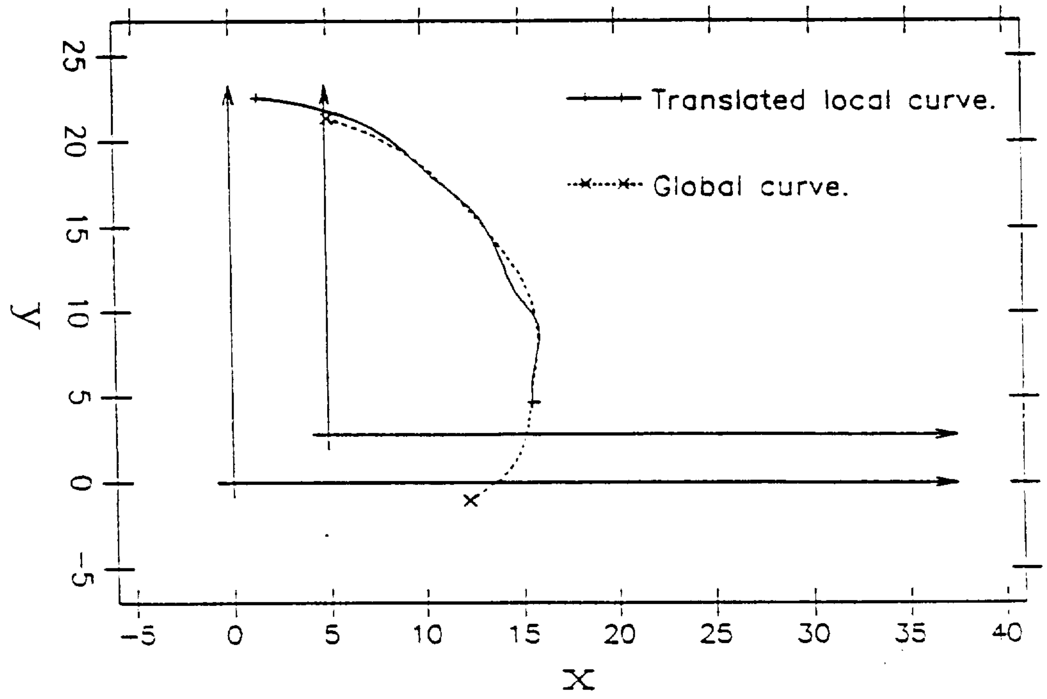

Figure 5.16: Curve match based on direction-to-intersect averaging for the noisy curves.

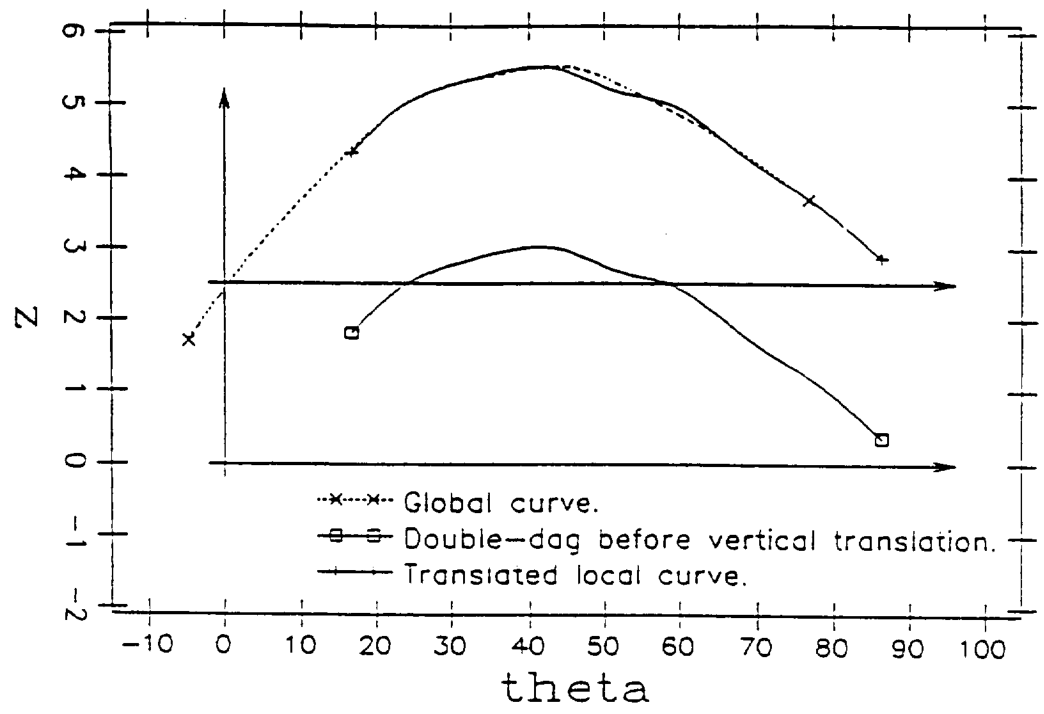

Figure 5.17: Matching of the vertical data for the noisy case, based on the correspondence developed from the horizontal data.

The terrain for the simulation, depicted in Figure 5.18, was the terrain described in Section 2.2— periodic in the x and y directions with a "monolith" superimposed onto it. The user specified a terrain matching problem to the program by entering the vehicle's actual vantage or "local vantage", and the vehicle's initial vantage estimate or "global vantage." The program then generated a set of local skylines from the terrain function, to simulate skylines collected from the sensors. The important distinction between local and global skylines, preserved in the simulations, was that the algorithm was given a single set of local skylines which it could not change, but it was free to generate global skylines for any test vantage above the terrain.

Given the estimated vantage and the local skylines, the algorithm iterated through the five steps enumerated in Section 4.3. As mentioned previously, a curve correspondence algorithm has not yet been implemented, and so each time the algorithm reached that step, it displayed the curves graphically on the workstation

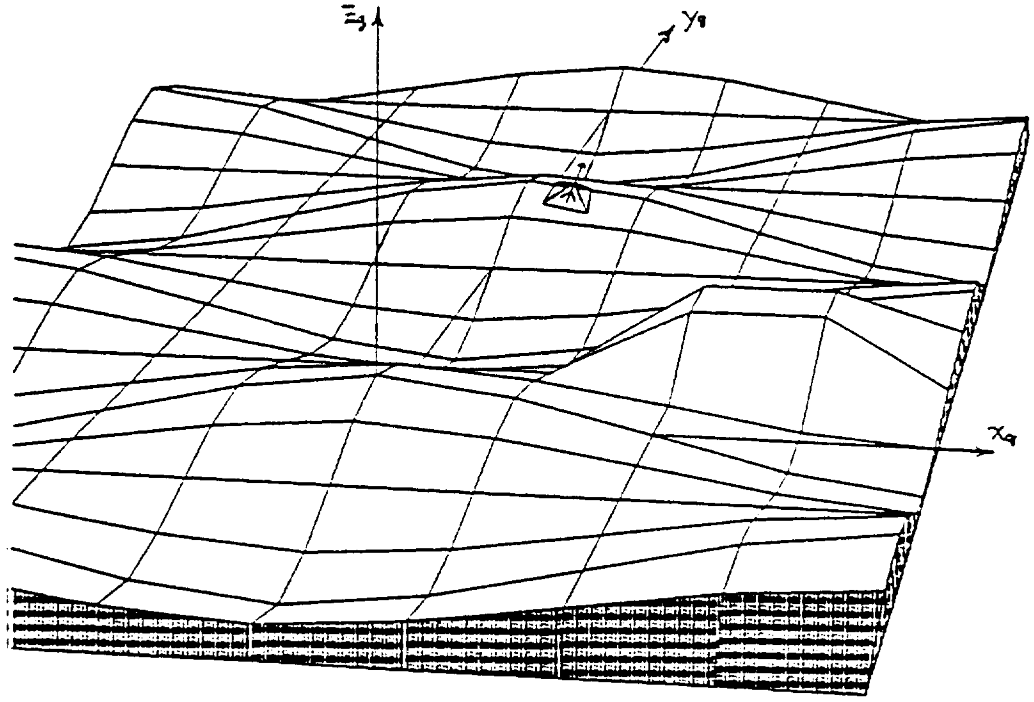

Figure 5.18: Example 1: the "true" location of the vehicle.

screen, and prompted the user to specify and/or edit the curve correspondence. The termination criterion for the algorithm was that the vantage estimate change by less than 0.1 meter in any single iteration.

In the experiments described below, the model of the rough map of the simulated terrain was bypassed during generation of the global skylines, and instead the skyline generation routines had direct access to the true terrain function. In other words, the terrain matching algorithm was effectively simulated using a perfect global map. All linear coordinate measurements are in meters.

Example 1 was the following problem: the vehicle's true vantage was (x_v, y_v, z_v) $= (2, 38, 6.43)$, with a yaw of 90°, or facing north. A graphical depiction of the vehicle at the true vantage is shown in Figure 5.18, and this vantage is two meters to the right, and two meters forward of the top of the hill in this view. (Vehicle is drawn with the actual pitch and roll for its position, but pitch and roll are neglected in the analysis. The vantage is assumed to be two meters above the terrain, regardless of

Iter.	$\hat{x}_v$	$\hat{y}_v$	$\hat{z}_v$	Remaining Error Magn.	Incremental Improvement	Overall Improvement
0	0.00	40.00	7.00	2.89		
1	1.43	38.35	6.27	0.69	76.1%	76.1%
2	1.48	38.12	6.38	0.54	22.4%	81.3%
3	2.13	37.82	6.51	0.23	56.5%	92.0%
4	2.10	37.95	6.44	0.11	52.4%	96.2%
5	2.09	37.96	6.43	0.10	11.8%	96.5%

Table 5.2: Progress in each iteration of Example 1. Actual vantage (2, 38, 6.43).

the gradient of the terrain on which the vehicle lies.) The terrain height at $x^g = 2$, $y^g = 38$ is $z^g = 4.43$, and the vantage is two meters above the terrain. The vehicle was given an initial vantage estimate at precisely two meters above the top of the hill, $(\hat{x}_v{}^0, \hat{y}_v{}^0, \hat{z}_v{}^0) = (0, 40, 7.00)$. Before calling the terrain matching algorithm, the program generated skylines from the actual terrain function for the actual vantage, to serve as the local skylines. The algorithm was given the local skylines in cylindrical coordinates about the local vantage, and the estimated vantage.

This example took five iterations before termination, and the effects of each iteration are recorded in Table 5.2. In the table, the error magnitude is the distance between the true vantage and the vantage estimate. Percentage improvements were calculated as [1 - (new error magn.)/(old error magn.)](100%). Figure 5.19 graphs the horizontal projection of the sequence of vantage estimates, as the algorithm grew closer to the true vantage. The dotted lines show the range of azimuth in which skylines were collected, which was ±60° from the yaw. Each 'x' represents one of the vantage estimates in Table 5.2. From its initial estimate of (0, 40, 7.00), the algorithm settled at an estimate of (2.09. 37.96, 6.43).

Figures 5.20 and 5.22 show the corresponded curves at the initial and final vantage estimates, illustrating the curve "matching" accomplished by the algorithm. Figures 5.20 and 5.22 show the horizontal projections of the curves, and Figures 5.21

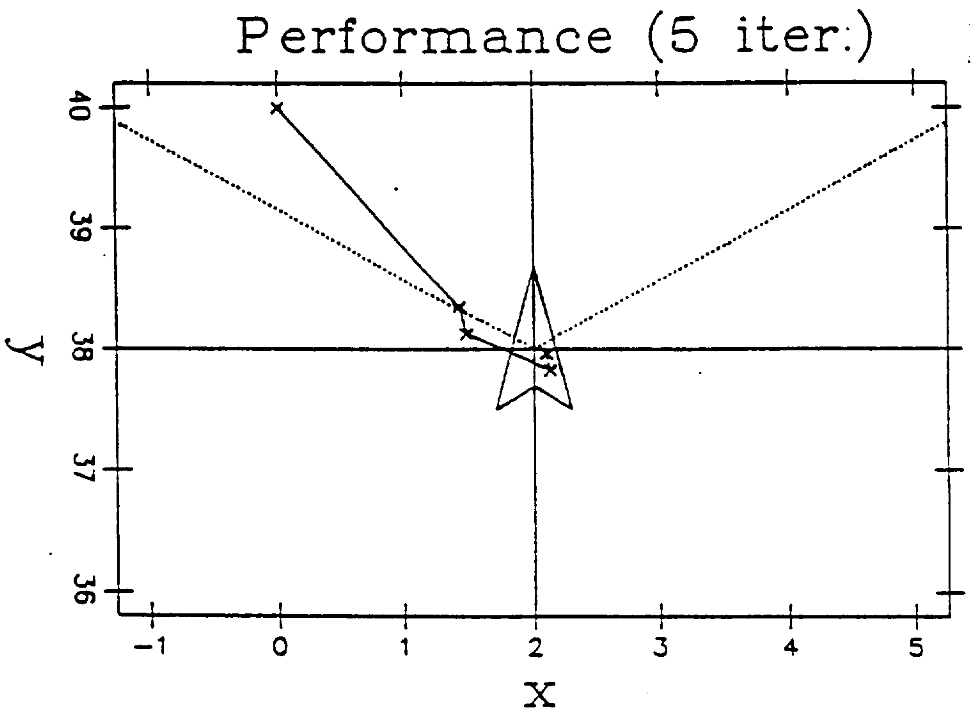

Figure 5.19: Example 1: sequence of vantage estimates.

and 5.23 show height ($z^* = z^\dagger$) vs. azimuth ($\theta^* = \theta^\dagger$). The solid lines indicate global skylines, and the dotted lines indicate local skylines. Only the corresponded curves are shown. Note the multiple correspondence in the initial sets, in which two global skylines corresponded to the same local skyline. Also note that a local skyline (at a horizontal ρ of about five meters) which did not have a corresponding global skyline at the initial estimate, did have one at the final estimate. The two sets of skylines grew more similar as the vantage estimate grew more accurate.

The correspondence shown in Figure 5.22 was performed for illustration only and was not part of the algorithm's normal execution. The algorithm terminated immediately after determining that the fifth iteration made a sufficiently small vantage update, and did not process the global skylines at its final vantage estimate. That is, these would be the corresponded curves of a hypothetical sixth iteration.

More figures in the form of Figures 5.19 and 5.20 are shown for Examples 2–4 in Figures 5.24–5.29, respectively. The algorithm's performance on the these examples

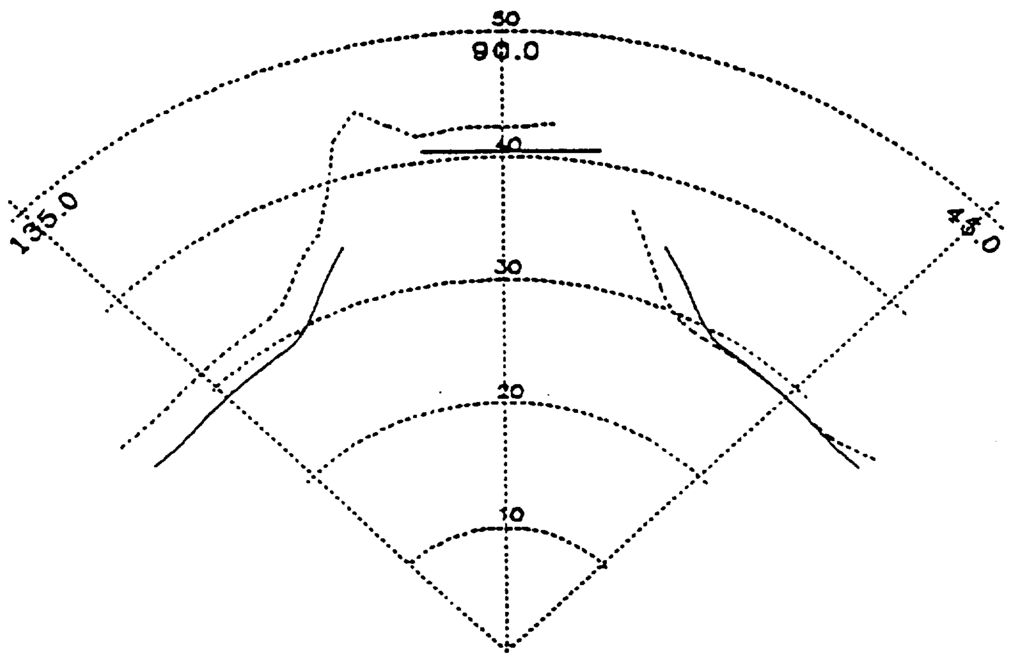

Figure 5.20: Example 1: corresponding curves for the initial estimate (horizontal data).

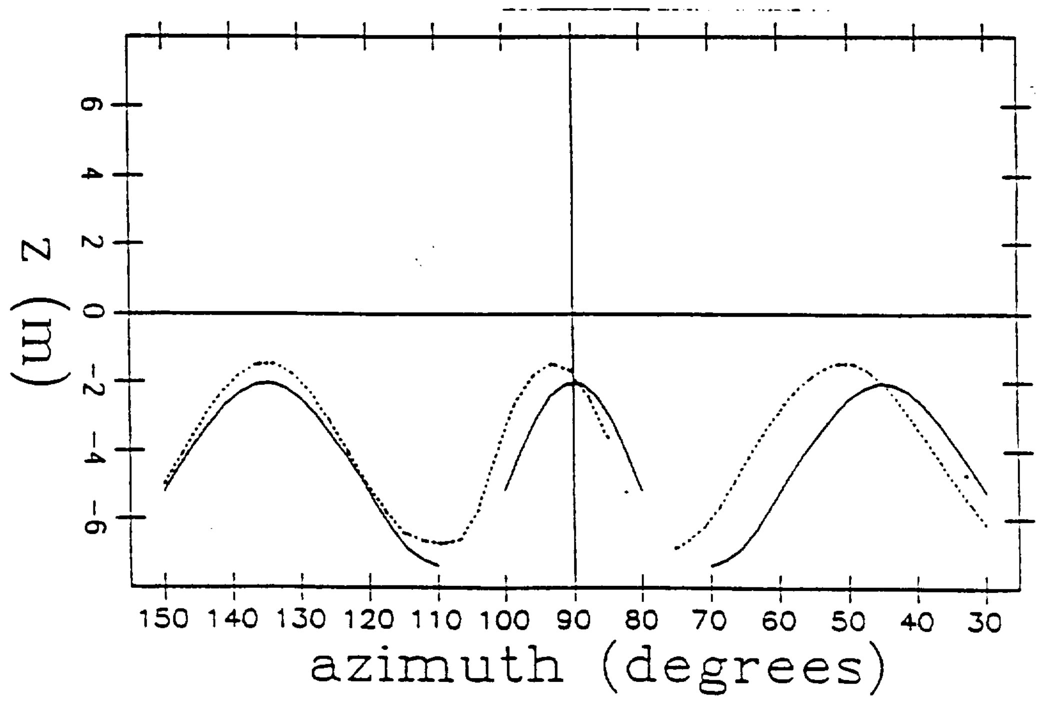

Figure 5.21: Example 1: corresponding curves for the initial estimate (vertical data).

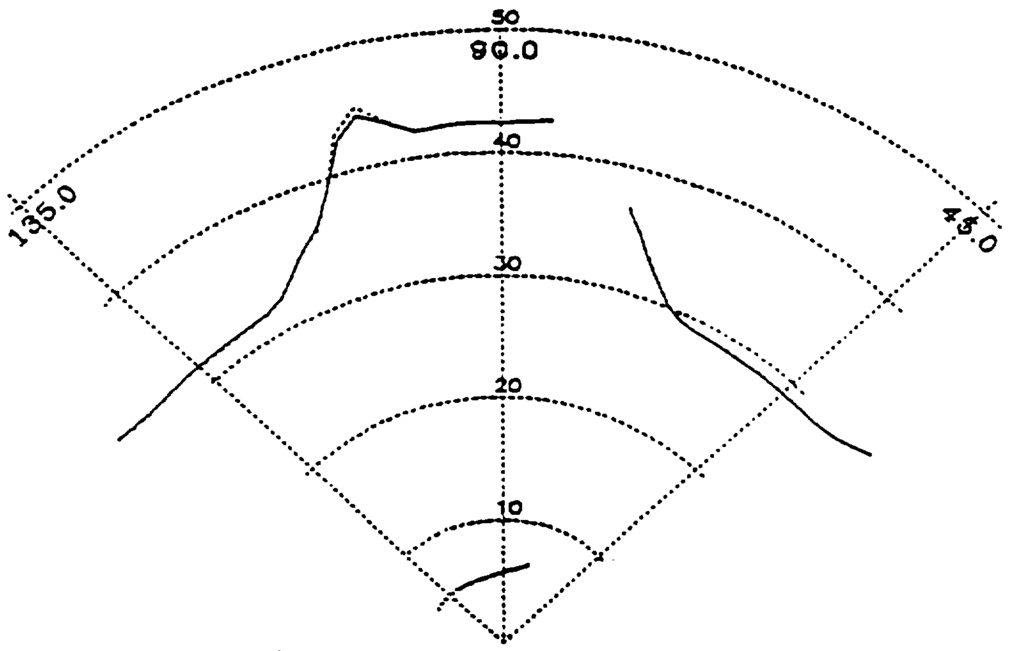

Figure 5.22: Example 1: corresponding curves for the final estimate (horizontal data).

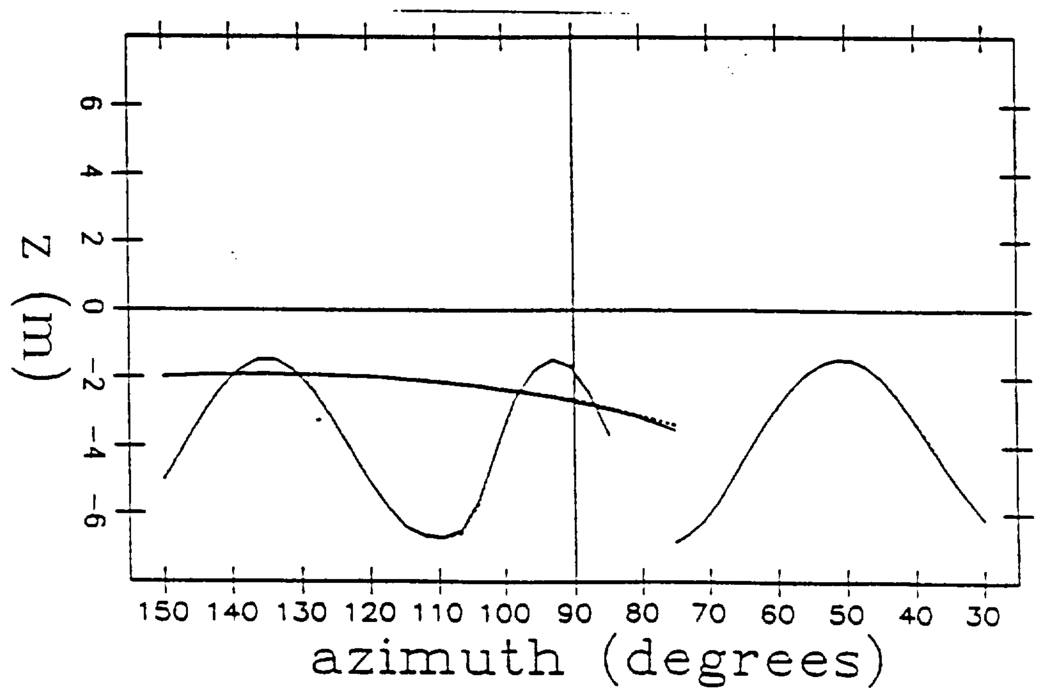

Figure 5.23: Example 1: corresponding curves for the final estimate (vertical data).

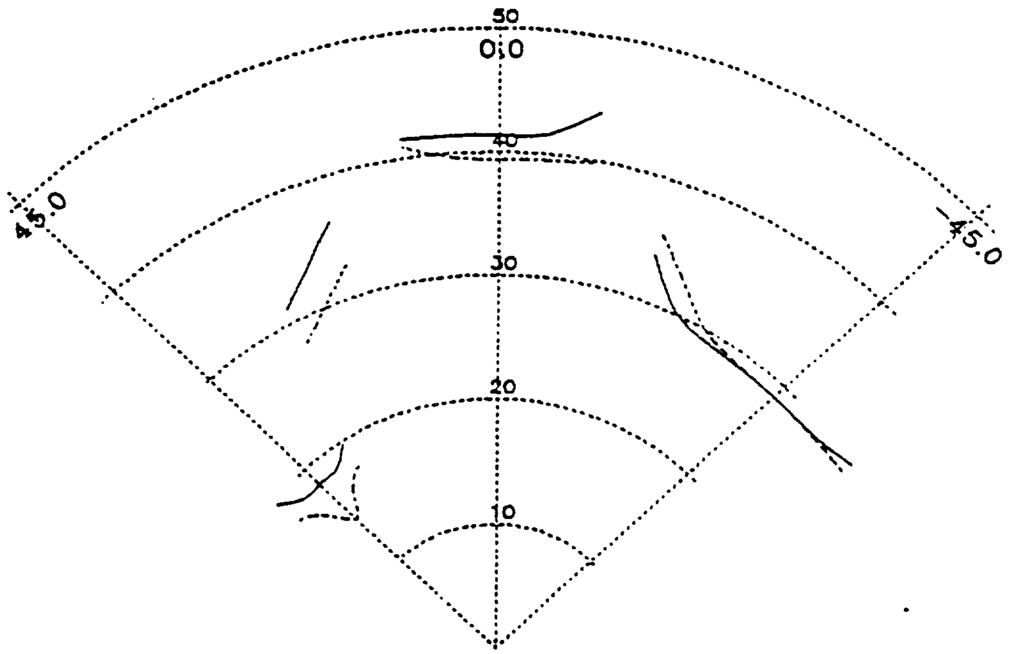

Figure 5.24: Example 2, horizontal curve data.

is summarized in Table 5.3. In Example 2, the vehicle faced a yaw of 0° (west), lying one meter north and one meter west of the origin (the vertical line coming out of the terrain in Figure 5.18; the initial estimate was one meter south and one meter east of the origin. In this example, the second iteration was particularly successful, even though the first iteration made only marginal progress (19.1%). The first iteration brought the vantage estimate close enough for an excellent match to exist in the second iteration.

Example 3 put the vehicle about ten meters north of the origin, at the steepest part of the descent in the valley, facing northwest. The bulk of the progress in this example was made in the first two iterations; the vantage estimates continued to improve at a consistent but less dramatic rate after that.

In Example 4, the vehicle was facing north, ascending the hill at the origin from the south. Once again, there were some iterations— in this case, particularly the second— which produced only marginal progress, but this progress was sufficient

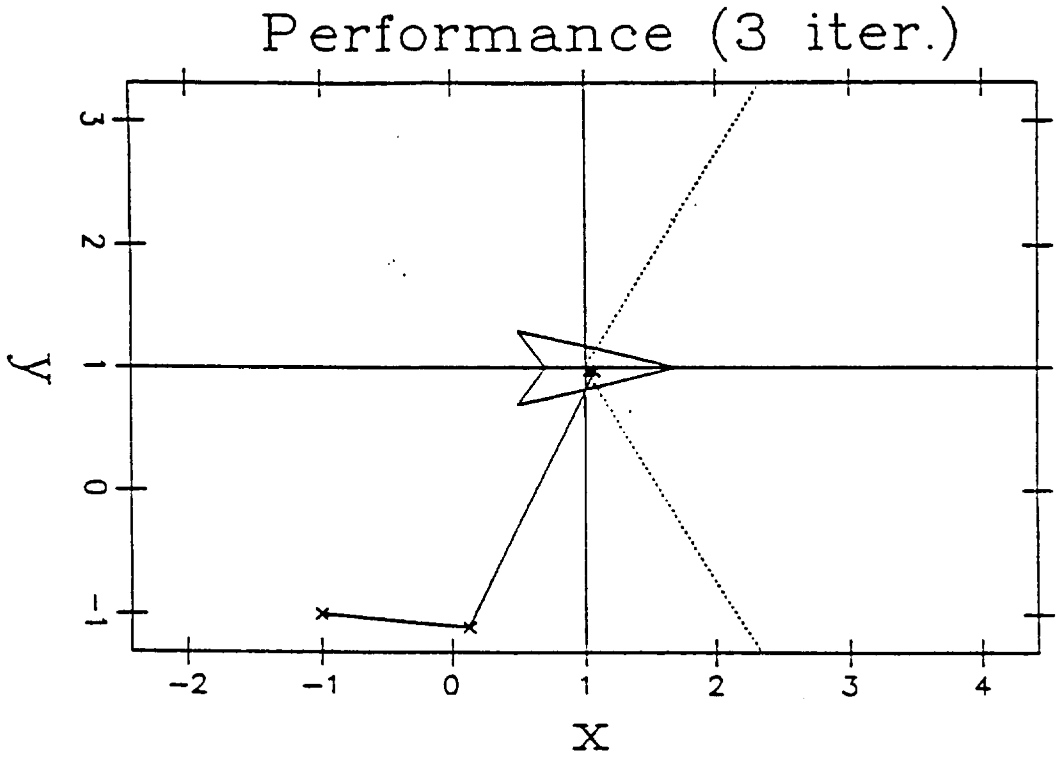

Figure 5.25: Example 2 results.

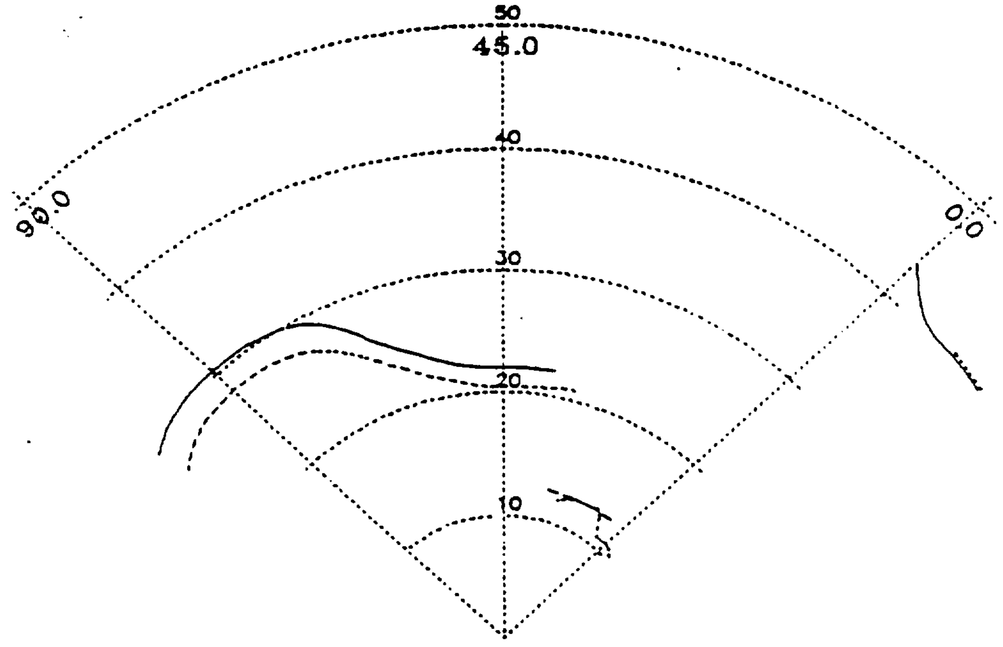

Figure 5.26: Example 3, horizontal curve data.

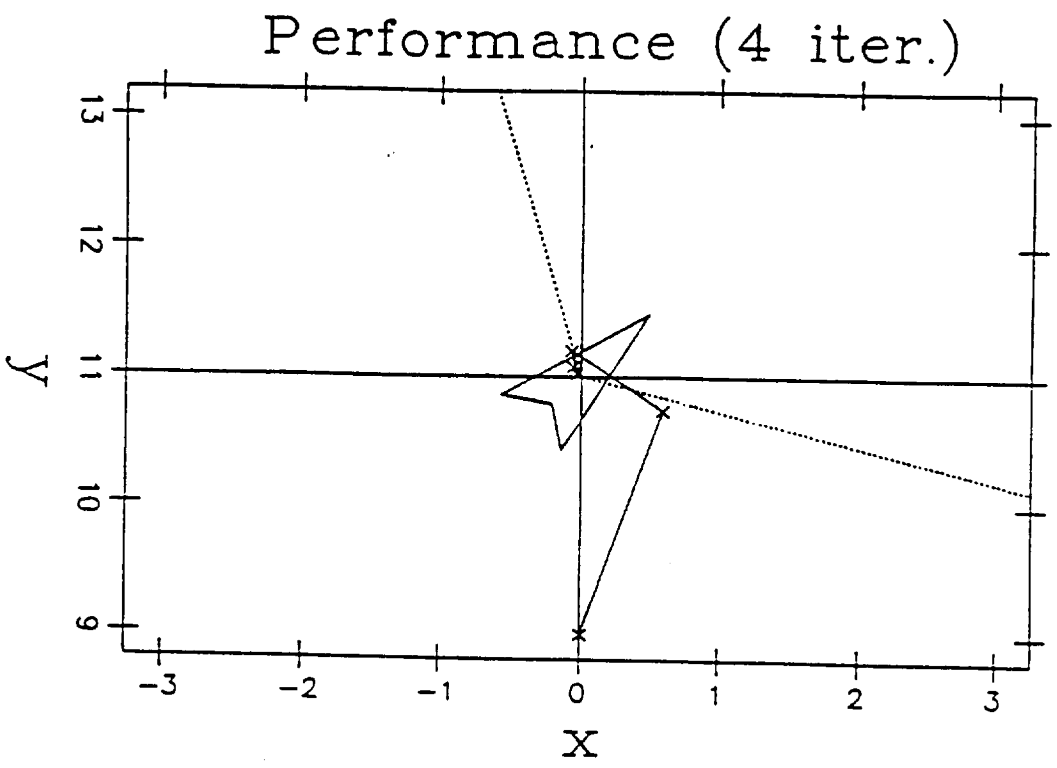

Figure 5.27: Example 3 results.

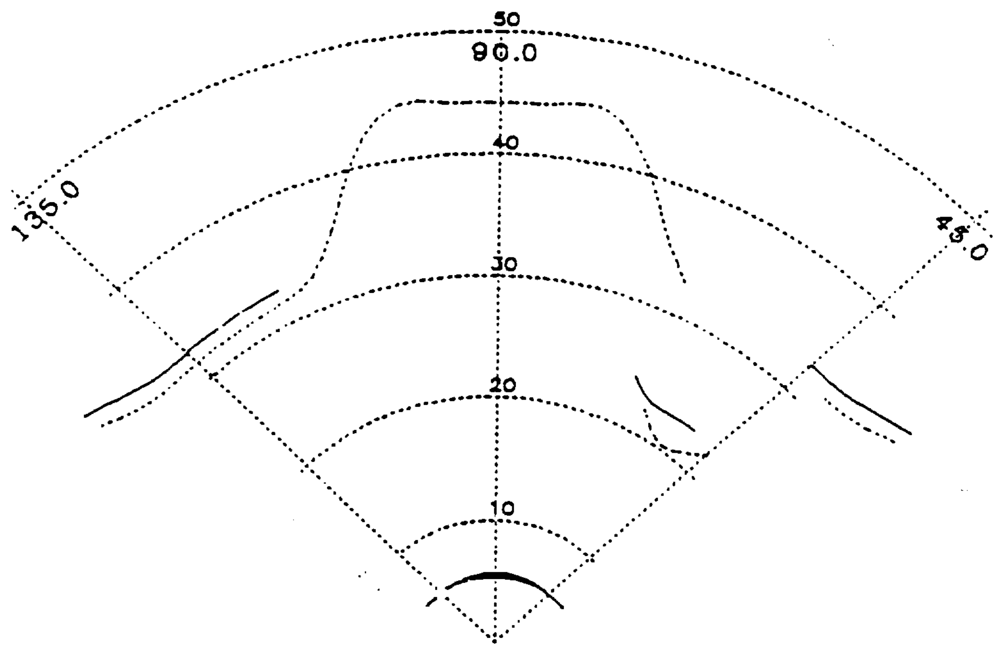

Figure 5.28: Example 4, horizontal curve data.

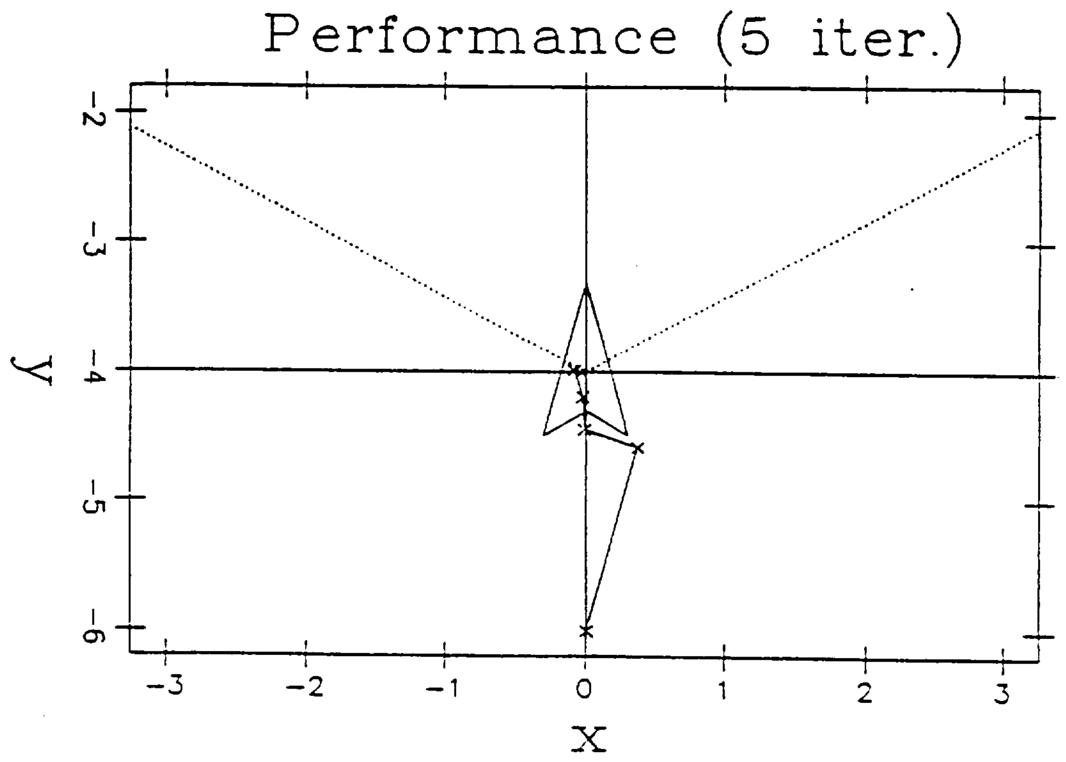

Figure 5.29: Example 4 results.

Example No.	Local (True) Vantage	Init. Vantage Estimate	Yaw	Init. Error	First Iter. Improvement	Iters. to Terminate	Final Error
1	(2., 38., 6.43)	(0., 40.0, 7.00)	90°	2.89	76.1%	5	0.10
2	(1., 1., 6.85)	(-1., -1., 6.85)	0°	2.83	19.3%	3	0.03
3	(0., 11., 1.43)	(0., 9., 2.57)	45°	2.30	70.3%	4	0.09
4	(0., -4., 5.86)	(0., -6., 4.63)	90°	2.35	70.5%	5	0.04

Table 5.3: Performance summary for Examples 1–4.

to facilitate better matching in successive iterations.

CHAPTER 6
CONCLUSION

The terrain matching problem has been described in the context of autonomous vehicle navigation over a $2\frac{1}{2}$-D unstructured terrain, such as the surface of Mars. It consists of finding the vantage point of the vehicle. Our approach to the problem makes use of "skylines," or occluding contours of the terrain which are visible from the vantage. The skylines, unlike boulders or other conventional "features," should be visible (or estimable) both from onsite and flyby observations. By comparing two versions of these features, from the sensors and from the map, terrain matching finds the relationship between their respective coordinate frames. In this work, the rotational relationship between the two coordinate frames is assumed known, and the terrain matching algorithm must update the translational vantage parameters, given an initial vantage estimate.

We approximate the two sets of skylines as identical, even though the actual set of skylines depends on the particular vantage from which they are seen, and thus an error in the vantage estimate causes the skylines to be slightly different. Given that the curves are the same, but are represented with respect to two different vantages— the true vantage and the estimated vantage— the representations of those curves with respect to their vantages will be related by the same translation (in the reverse direction). The problem is thus reduced to finding a translation between the two sets of curves, when their vantages have been overlaid onto each other. We refer to this as translating the locally seen skylines into the "dagger" coordinate system, which is coincident with the coordinate system in which the skylines from the global map are represented.

The search for the translation between the curves is based on a "Max-min Principle," even though exceptions are possible when using realistic curves that have noncorresponding endpoints. We use normal-to-intersect calculations in place of shortest-distance measurements to simplify the computation. The longest of these is chosen as the max-min. The max-min's direction estimates the direction of the horizontal translation between the curves, and the average direction-to-intersect distance between the curves in that direction estimates the length of the horizontal translation. The vertical component of the translation is also found by using the max-min's direction as a rule of point-by-point correspondence, and averaging the vertical distances between corresponding curves.

Each iteration of the algorithm generates a new estimate for the vehicle's vantage point, by adding the estimated 3-D translation between the skylines to the previous estimate. In successive iterations, the local and global skylines grow more similar as the vantage estimate grows more accurate.

The method introduced here lies in the category of "feature matching" as opposed to "iconic matching," in the terminology of [11]. The advantage of feature matching is that particular data is extracted from the two sources of information (the local and global maps) prior to comparing the data, rather than matching the entire maps to each other directly. The compuatational complexity is reduced, because only a small subset of the available data is processed in detail. We must acknowledge, however, that the computational advantage of using feature-based matching is blunted by the fact that extracting the feature data, that is, generating the skylines from the rough map, is relatively expensive.

Our method also has a fundamental distinction from the "cost function" approaches, for instance [13] and [3, sect. 5]. These evaluate a cost function representing the "quality of match" between the two sets of data in a grid of trial vantages, and update the vantage based on those samples of the cost function. Although the method introduced here also requires repeated iterations of vantage estimation when applied to noisy skylines, each iteration updates the vantage estimate based on only a single prior estimate, and not on a whole grid of trial vantages.

Except for the curve correspondence step, the algorithm has been successfully implemented for simulation by C language computer programs. The algorithm has performed well on a variety of test cases, such as the examples given in Chapter 6. These examples demonstrate that even iterations which provide marginal improvement are valuable, because the resulting skylines at the new vantage estimate match the local skylines even better.

The algorithm's computational efficiency has been difficult to evaluate because the simulations are inherently graphics-intensive. Since the curve correspondence is performed by the user at run-time, a large amount of intermediate skyline data has to be displayed which would otherwise not be necessary. The graphics generation puts a substantial "overhead" burden on the current programs, but the algorithm's cycle time is approximately 10 to 14 seconds on a Sun 3/60. This duration includes the generation of skylines from the rough map but does not include the time spent waiting for the user to edit the curve correspondence.

A limitation of skyline-based terrain matching is the need for enough skylines to be visible from a vantage to contain the required 3-D information. This depends on two factors: first, the azimuth range of skyline collection by the sensors and the rough map ($\pm 60^\circ$ in our examples) must be wide enough. A good max-min depends upon at least one corresponded skyline to produce a normal-to-intersect measurement in the direction of the translation between the curves. Increasing the azimuth range of skyline collection increases the chances of some skylines having normals in the direction of the actual translational error. Second, it is up to the particular terrain on which the vehicle lies to provide adequate skylines for the algorithm to collect and use. A terrain which is simply too flat and non-descript will not provide enough skylines for reference. We can control the first factor, but the second depends solely on the environment in which the algorithm is applied.

In its current formulation, the algorithm is somewhat susceptible to errors in the max-min by simply taking the single biggest normal-to-intersect. As the azimuth width of skyline collection is increased beyond about $\pm 75°$, certain anomalies (described in 4.2.1) become more common because of long, twisty skylines with multiple correspondences which arise. Such skylines contain a great amount of information, but also a lot of ways in which strange relationships may appear.

One way to improve the algorithm's robustness in this regard would be to insert a test for each normal-to-intersect, making sure sure that it represents a local minimum and not a maximum distance, from its terminating point to kits base curve.

LITERATURE CITED

[1] Brian Wilcox and Donald Gennery. A mars rover for the 1990's. *Journal of the British Interplanetary Society*, 40:484–488, 1987.

[2] C. N. Shen and George Nagy. Autonomous navigation to provide long-distance surface traverses for mars rover sample return mission. In *Proc. Fourth IEEE International Symposium on Intelligent Control*, pages 362–367, Albany, NY, September 1989.

[3] C. N. Shen and Lance Page. Fusion of gross satellite sensing and laser measurements by skyline map matching for autonomous unmanned vehicle navigation. In *Proc. SPIE Optical Engineering and Aerospace Sensing Symposium*, Orlando, Florida, April 1990.

[4] C. N. Shen and Lance Page. Skyline-based terrain matching using a max-min principle. Technical report, NASA Center for Intelligent Robotic Systems for Space Exploration, Rensselaer Polytechnic Institute, October 1990. CIRSSE Report #72.

[5] Lance Page and C. N. Shen. Analysis of terrain map matching using multi-sensing techniques for autonomous vehicle navigation. In *SPIE Symp. on Advances in Intelligent Systems*, Boston, Massachusetts, November 1990.

[6] J. C. Mankins (ed.). Mars rover technology workshop proceedings. Technical report, Jet Propulsion Laboratory, April 1987. JPL D-4788.

[7] C. N. Shen and George Nagy. Autonomous terrain navigation using a visibility-oriented digital terrain model. In *SIAM Conference on Geometric Design*, pages 362–367, Tempe, Arizona, November 1989.

[8] Eric Krotkov. Mobile robot localization using a single image. In *Proc. IEEE Robotics and Automation Conference*, pages 978–983, 1989.

[9] C. Ming Wang. Location estimation and uncertainty analysis for mobile robots. In *Proc. IEEE Robotics and Automation Conference*, pages 1230–1235, 1988.

[10] Paul R. Klarer. Autonomous land navigation in a structured environment. *IEEE Aerospace and Electronic Systems Magazine*, 5(3):9–11, March 1990.

[11] Hebert, Kanade, and Kewen. 3-D vision techniques for autonomous vehicles. Technical report, Carnegie Mellon University, August 1988. CMU-RI-TR-88-12.

[12] Fu, Gonzalez, and Lee. *Robotics: Control, Sensing, Vision, and Intelligence.* McGraw-Hill, 1987.

[13] Donald Gennery. Visual terrain matching for a mars rover. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Diego, California, June 1989.

[14] Long Quan and Roger Mohr. Matching perspective images using geometric constraints and perceptual grouping. In *Proc. IEEE*, pages 679–684, 1988.

[15] Jian-Der Lee, Jau-Yien Lee, Chin-Hsing Chen, and York-Yih Sun. A new approach to robot guidance in an unfamiliar environment using indication-post. In *Proc. IEEE Robotics and Automation Conference*, pages 1433–1438, 1989.

[16] Olivier D. Faugeras Nicholas Ayache. Building, registrating, and fusing noisy visual maps. *International Journal of Robotics Research*, 7(6):45–65, December 1989.

[17] M. Hebert, C. Caillas, E. Krotkov, I. S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *Proc. IEEE Robotics and Automation Conference*, pages 997–1002, 1989.

[18] William H. Press, Brian R. Flannery, Saul A. Taukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1988.

APPENDIX A
SOFTWARE SUMMARY: PROGRAMS AND MODULES

The terrain matching software package consists of three programs, and several "modules" which constitute the building blocks for these programs and other possible terrain matching programs in the future.

The three main programs are:

`t7` — "Terrain program, version 7". Interactive program to demonstrate skyline generation for the simulated terrain. (Generated the examples shown in Section 3.4.)

`mm4`— "Max-min demonstration, version 4." Graphically demonstrates the steps of the terrain matching algorithm for a single pair of curves.· Command-line argument may be used to specify noise injected into the data. (Generated the results presented in Section 5.1.)

`cm6`— "Curve matching, version 6." Demonstrates the whole terrain matching algorithm except for curve correspondence which is performed by the user (in an interactive, graphical environment). Also supports other interactive modes for skyline generation and display. (Generated the results presented in Section 5.2.)

An additional program is `t5`, a predecessor of `t7`. It has a somewhat more primitive and less modular implementation than `t7`, but includes a "demonstration" mode which graphically shows individual steps in the skyline collection.

The main analytical modules are:

`sg`— "Skyline generation module." Skyline generation as described in Chapter 3, from either the global map or the actual terrain function.

`mm`— "Max-min based terrain matching module." Implements the algorithms of Chapter 4 for computing the max-min, and for performing the direction-to-intersect averaging. Algorithm is not complete, however— curve correspondence is not implemented.

The modules for supporting graphics output are:

`plot`— "Plot module." Provides basic graphics support, and provides a reasonably convenient utility for 2-D data plots, including:

- Rectangular and polar modes.
- Bar plots.
- Axis drawing.
- Labeling of axes and data.

Implementation is built upon the `SunCore` graphics library.

`tg1`— "Terrain graphics, module 1." Provides an "aerial view" of the terrain, and of a "rover" on top of the terrain.

Additional support modules are:

`ter`— "Terrain module." The terrain model described in Section 2.2.

`map`— "Map module." The map model described in Section 2.3.

`m4`— "4-dimensional math module." Basic math utilities, especially 4-D vectors and matrices.

`point`— "Point module." Provides a data structure for 3-D "points," in rectangular, cylindrical, or spherical coordinates. Supports linked lists of points, linked lists of linked lists of points, and conversions between the three coordinate systems.

`info`— "Information window module." (Used only by program `t5`— essentially "phased out.") Provides a "help and information" window, based upon the `plot` module and the `SunCore` graphics library.